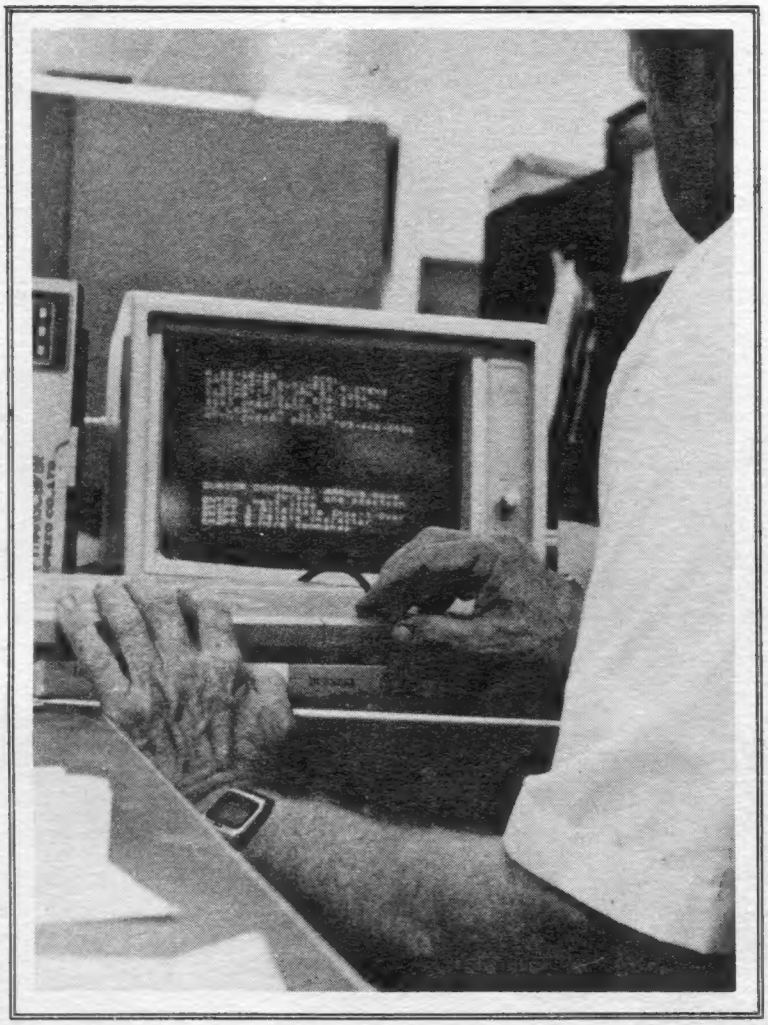
PRO/FILE 2068



The Associative Data Base

Thomas B. Woods

Pro/File 2068

The Associative Data Base

by

Thomas B. Woods

Published by
Thomas B. Woods
P.O. Box 64, Jefferson, NH 03583

This program is dedicated to all TS2068 owners who have kept faith in their ill-fated computer and who, despite all odds, are slowly managing to figure out how the blasted thing works.

ACKNOWLEDGEMENTS

I wish to thank several individuals who helped in the production of this book. Stephen Page wrote much of the "How to Use" chapter. Also Dennis Krill, member of the Central Pennsylvania ZX/TS User Group, spent many hours using the program and writing most of the "Applications" section. Oscar Sensabaugh, historian, geneologist, Timex computer fanatic, and great American humorist, wrote how to use Pro/File 2068 as a geneological record keeper. Robert D. Hartung, a fellow Timex enthusiast and frequent author in SYNC Magazine, helped in reviewing the "Modifications" chapter of this book to insure that no errors in the instructions were present. Last, and definately not least, thanks goes to my wife, Adele, for proof-reading the entire manuscript with the exception of this page. My spelling and punctuation skills leave much to be desired. Adele didn't know what she was getting into when she said she'd read the manuscrupt.

Thanks,

Tom Woods

Contents

INTRODUCTION: WHAT IS PRO/FILE 2068?

Pro/File 2068 is a data base program for the TS2068. You can think of it as an electronic box of index cards. With real index cards, you write the information you want to keep on them. Then you organize the cards in some convenient order so when you need to look something up, you can find the right card with a minimum of hassle. Cards holding a list of names, say, could be organized alphabetically so that when you need to find someone named SMITH you flip to the names beginning with "S". Then you hunt until you find the card that says SMITH.

It Stores....

With Pro/File 2068 you replace the index cards with the computer. The TS2068 serves as the box which holds all your cards. Its keyboard replaces the pencil you used to add or alter information on the cards. One big difference is that when you add new "cards" to Pro/File 2068, the computer DOES NOT place them in any particular order. Instead, new cards simply go to the back of the box.

It Searches....

The real index card system works great. It is a real time saver. So why would you want to go to the trouble of doing the same thing with a computer? For starters, the computer can find a particular index card faster, with more precision, and with greater ease than you can using a file card box. This may not seem logical since "cards" (read that FILES) are not stored in any order by the computer. In truth, the computer doesn't care if files are in order or not. Ordering to save time when looking for a specific file is pointless because the computer can search through EVERYTHING in less time than it would take you to lift the lid of the card holder.

Think for a minute about what you do when you want to look up a name using a real file card system. If you want to find the card that has Mr. SMITH listed, the first thing (which you just did) is to define what it is you want to find--in this case, the word SMITH. Second, you scan through all the cards until you find one that has a name which matches the one you're hunting for. If all your cards are randomly stuffed into the box it might take quite a while to find the card with SMITH on it because you would need to look at every card in the box.

Alphabetizing the cards saves time because it eliminates the need to search through everything. Knowledge of the alphabet tells you that the letter "S" is near the end so you go straight to the back of the box. Once you find the cards that start with "S", you start looking for the word SMITH. You stop looking when you find a card which matches the word you are searching for. This line of thinking may seem rather dull and elementary, but if you want to understand how Pro/File 2068 works and thereby make the most of all its capabilities, it helps to understand how you would do the same job "by hand."

Pro/File finds files the same way you would if your card holder held cards in random order. You define the word you want to find by typing it in on the keyboard. Throughout this book, this procedure is called "entering a SEARCH COMMAND." After you type a search command, you press the ENTER key. This sets the computer off and running. It scans every file you have stored in the program. First it hunts for "S." When it finds one, the program matches the next letter or character in the file with the next letter in the search command (the letter "M"). As letters match, the procedure continues until the entire word "SMITH" is found. Then the computer stops matching and displays the individual file (the card) on the TV screen. All this happens in the blink of an eye.

And it Searches Some More....

Pro/File 2068 carries this idea of "search and find" several steps further. Besides just being able to search for one word like SMITH, it can also search for two, three, or even more completely separate words. To draw an analogy using file cards, this is like having cards that have names, birthdays, and musical instruments on them. The "MULTI-WORD SEARCH" would be like finding everyone named SMITH whose birthday is in APRIL and who can play the TRUMPET. Three words--SMITH, APRIL, and TRUMPET form the search command. Pro/File can scan the data (or files) stored in the program and pull out only those that hold matches to word 1, AND word 2, AND word 3. Smiths born in June, or whose instrument is listed as the tuba are not displayed.

Multi-word search capability gives Pro/File 2068 a definite edge over the file card box. As the number of words used in a search command increases, it becomes more difficult to alphabetize the file cards so as to save search time using the card holder. In fact, the job of organizing the fifty SMITHs you may have into subgroups ordered by birthday and further by musical instrument, becomes a downright pain in the neck. The notion of just throwing the whole mess into Pro/File becomes very appealing. You can still have the quick searches and you don't need to bother keeping everything in order. You can have your cake and eat it too!

It ORGANIZES What it Finds....

"But I kinda liked the idea of alphabetizing", you say. RESPONSE: No problem! Pro/File 2068 can order in any of a number of ways. Suppose you add a batch of data to the program (how to do this will be covered in a bit) which is roughly equivalent to the cards holding names, birthdays, and instruments. You can search for any group of individual records (or files) and order each of them either by name, birthday, or type of instrument played.

This beats the file card box hands down. It is impossible to order cards in more than one way at the same time. You could have every card alphabetized first by name. Then for every card that has the name SMITH, you could further order them by birthday. You could even go to the extreme and order every SMITH born in JUNE still further by instrument. But all this effort wouldn't help much if you want to find all people born in JUNE ordered alphabetically by what they play in the band. With Pro/File 2068 you can do it. First you enter your search command for "JUNE". Then you tell the computer to order every file it finds so that "instrument played" will appear in alphabetic order.

It Caters to Your Every Whim....

To accomplish this feat with a file card box, you would have to reorder every card so birthdays were the primary basis of ordering. Then you would need to suborder every month by musical instrument. Names would not be important.

If you ask your secretary to reorder your file box everytime you wish to find things in a slightly different format, you will soon be regarded as a tyrant. You might even get the whole box dumped upside-down on your desk if you make this demand twice in the same day.

If you find yourself in this situation, now is the time to put all those cards into Pro/File. The computer doesn't care how many times you change your mind. It will give you whatever you ask for whenever you want it. Your secretary NEEDS Pro/File 2068.

And Once it Finds a File....
it asks what you want to do with it.

At this point the secretary who does not have Pro/File is probably already thinking up some good answers to this question. Pro/File 2068 is concerned with only two: Do you want the files that are found displayed just on the TV screen, or do you also want them printed out on paper?

If you tell the computer, "Yes, run the files through the printer," you can further specify which portions of each file you want to have printed out. This means that you could put ADDRESSES along with the names, birthdays, and musical instruments. When June rolls around, and you wish to send a note to every member of your "Slurp-n-Burp" dinner and band ensemble that this month is SMITH's birthday and everyone should come early for the party, you can make Pro/File 2068 print out just the names and addresses on labels. All the other information in each file (birthday and instrument) will be skipped.

Another time when you're putting together the program for your next concert, you could print out just the member's names and the instruments they play, leaving out the addresses and birthdays. The list you make could be alphabetically ordered by name, by instrument, or even by some other piece of information which is not printed out on paper. Pro/File is a pretty versatile program.

YEAH, BUT WHAT'S IT GOOD FOR?

Anything you can put in a box of file cards can be accessed better in Pro/File 2068. If you're like me and you never could get into using file cards, some things which come to mind include mailing lists, appointments, abstracts of magazine articles, church directories, club memberships, research notes, historical records, business transactions, inventories, price sheets, catalog descriptions, ham radio logs, reference material like conversion tables and part specifications, records of mail-order purchases.... The list goes on.

A substantial chapter of this book describes in concrete terms just how you might implement some of these applications. These are intended to give you ideas. The more you use the program, the more possibilities will occur to you.

HOW DOES PRO/FILE DIFFER FROM OTHER DATA BASE PROGRAMS?

What makes Pro/File 2068 different from other computer data base programs? There are some very important differences. Every other "file manager" that I have seen takes the information you give it and categorizes it into subgroups called "FIELDS." The NAMES, BIRTHDAYS, and MUSICAL INSTRUMENTS used in the previous illustration could be thought of as "fields". There is a "Name" field where you type in a name, a "Birthday" field which holds the birthday, and an "Instrument" field which stores the type of musical instrument played by the person listed in the name field. The three fields taken together are called a "RECORD" which is held by the computer along with many other records all containing the same number of fields.

Before you ever type a single bit of data into another file program, you must first define the number of fields you wish to use and in many programs you must also specify the length of each field. In other words, first you decide that there are going to be three fields: NAMES, BIRTHDAYS, and INSTRUMENT. Second, you have to guess how long your data is going to be that goes into each field. Some names like Joe Blow consist of only 8 spaces. Other names like Joseph Preston Anderson III require 27 spaces. The number of spaces required by each field is the "length" you specify. Naturally you want your fields to be long enough to hold the longest name you have.

Herein lies a disadvantage. If you define field lengths long enough to handle your longest entries, all the Joe Blows you add will waste memory space. Field lengths are etched in granite. If your data base can hold 300 Joseph Preston Anderson III's, it will likewise hold only 300 Joe Blows even though Joe's name only takes up a third the space of Mr. Anderson.

The NUMBER of fields is also written in granite. Once you define the number of fields, you're out of luck if you later decide to add a new one. You can play it safe and specify a few extras to use if you change your mind, but like field lengths, unused fields waste memory space. There is another disadvantage when you must define fields. All the records you put into the computer must be of the same type. You can't put your appointment file in with your club membership. They must be kept in separate programs each with fields customized for its particular use.

For anyone who uses a TS2068 with a cassette recorder to load programs, the tedious and troublesome job of loading a cassette every few minutes just about kills the notion of making the computer a useful tool for storing and retrieving information.

Pro/File 2068 is different. You don't have to define fields. The computer doesn't know what they are. Instead, you have a "screen display" which can be 15 lines by 32 columns maximum. This display is your "index card" on which you can write just one word, a paragraph of text, a table of numbers, whatever you want. One screen display is the equivalent of one "record" using the old fashioned field structured data base. Throughout this book a screen display will be referred to as a "file" or a "record". The total collection of screen displays which is what you load into your computer will be referred to as "files" or "records".

A file can use any number of lines of a screen display, and each line can vary in length. When it is stored away in the computer, it goes into a large chunk of memory that simply takes what comes--long, short, it really doesn't matter. The only important factor is the total amount of space used for all the files. Pro/File has space for 28,000 letters, figures, spaces,or punctuation marks (all these being known as "characters").

The advantage to this system of storage is two fold. First, it means that you can store different types of files in the same program. You can keep your appointment calendar right in with your telephone directory and inventory listing. You can load a cassette once in the morning to get at all the data you need. Second, a short file does not waste memory space like it would in a field oriented program. More efficient use of memory means that you can store more information in the same space.

Sometimes, Pro/File 2068's way of taking information is called "Free form" or "Format free" data storage. In a sense, this is true. The computer lets you put whatever you want wherever you want. Don't make the mistaken assumption, though, that there is no advantage to "organizing" or "formatting" a file so that in each one, the same type of data will appear in the same place. The more method you apply to your madness, the more useful the program becomes. This becomes apparent as you read through this book. Many examples are given.

Determining the right mix of method to madness is probably the most confusing part of learning how to use Pro/File. It is a very individual matter that depends primarily on your personality and what you want to store in the program. The thing to do is read this book and try out ideas of your own. Pro/File gives you a tremendous amount of freedom in the way you store and access your files. At first, this freedom can be confusing but you'll come to appreciate the latitude you have as you become familiar with running the program. Don't be afraid to experiment. With Pro/File, it is relatively easy to change things around if you discover a new trick for storing or retrieving information. Even mistakes will teach you something new about how to use the program.

HOW TO USE PRO/FILE 2068

This chapter concerns itself with the mechanics of getting the program into the computer, filing information, and how to get it out again. There is a great distinction between how to do it and how to make the most of it. Writing into Pro/File 2068 is just like writing on paper. There are many ways to say the same thing. Some are a whole lot more informative than others. Putting data into the program is one thing. Doing it creatively so Pro/File 2068 can reveal subtle insights on the data you have stored is another. Here, the HOW TO is covered. Further on in the book the WHY you choose to do it in a certain way is discussed.

LOADING THE Pro/File 2068 CASSETTE

To put Pro/File 2068 in the computer, put the original cassette in your recorder. Make the necessary connections to the computer. Those unfamiliar with the proper procedure should READ THE BOOK THAT CAME WITH THE COMPUTER. After everything is set and the computer is powered on, type:

LOAD "pro/file" or LOAD " "

Press PLAY on the recorder and hit the ENTER key. The first thing that should come on the screen is:

Program: pro/file

After a few seconds, the screen will turn black and a new display will come up. Here, the copyright notice and a flashing message, "LOADING... please wait" will appear. After about another minute, Pro/File 2068 will finish loading and new instructions printed at the bottom of the screen ask you to:

PRESS "C" TO CREATE A NEW FILE
or "L" TO LOAD AN EXISTING ONE

At this point the program is ready to roll. What actually happened during this loading procedure was that the computer loaded three separate programs. The first one you loaded by entering the LOAD "pro/file" command. This program took over control of the computer and printed the "please wait" sign on the screen and loaded the remaining two programs which were on the cassette. The first of these was Pro/File 2068's machine code, a vital portion of the total. Finally, the Basic segment of Pro/File 2068 was taken from the tape.

A full explanation of what these three programs do and how they work together is covered later. For now, suffice it to say that in order for Pro/File 2068 to function, all three programs MUST be loaded into the computer.

LOADING PROBLEMS

A lot can go wrong between pressing PLAY on the recorder and seeing the "C-to CREATE, L-to LOAD" message on the screen. If you find that the master tape is difficult to load reliably, ask yourself:

> Do I have similar problems when I try to load tapes that I have made myself?

If the answer is yes, you have problems with either your tape recorder, the computer itself, or the cables that attach the two together. If you find you can load tapes which were made with the same recorder--even after the computer has been on for several hours--but Pro/File 2068 refuses to load no matter how you set the volume, two things could be wrong:

A) The tape is defective (possible, but not very likely)
B) There is a difference in the tape head alignment between your equipment doing the loading and my equipment which made the tape.

Tapes that do not load because of alignment problems are not defective. There are several tricks you can try to coerce Pro/File 2068 into your computer.

Play the Pro/File 2068 tape in your tape recorder and listen to it. Compare it against programs you made yourself. If the Pro/File 2068 tape sounds muddy, dull, and not as loud as your own tapes, and the high frequency tones do not come through on Pro/File 2068 like they do on yours, chances are very good that you have narrowed the loading problem down to tape head alignment differences.

Try using a different tape recorder to load Pro/File 2068. No two recorders are alike. You might find that one borrowed from a friend will work perfectly.

Another thing you can try is to readjust your tape head. Almost every recorder on the market has a small hole or notch in the top of the case. Location of this hole varies with the brand of recorder, but it usually is directly over (and a bit left of center) the tape head when the head is in the PLAY position. When you probe around in this hole with a tiny jeweler's screw driver, you'll find a screw which will change the angle at which the tape head meets the tape when the recorder is running.

To adjust the alignment, slowly turn the screw in either direction while you listen to the Pro/File 2068 tape. You won't need to turn more than a half turn which ever way you go. As you do you will hear a very noticeable change in sound quality of the tape. You want to aim for the loudest, crispest, highest pitched sound you can get. Unless you have a deafness for high frequency sound, the proper adjustment will be unmistakable.

Many people are reluctant to try this adjustment for fear they will open a pandora's box of future loading problems. For a long time I felt the same way until once in desperation I tried altering the head alignment of my recorder. I discovered that all of my fears were completely unfounded. In fact, a jeweler's screwdriver now takes a regular place beside my tape recorder. Any time I have trouble loading a program, I give the heads a little twist and in it goes. I have never purchased a program I could not load after adjusting my tapehead to match the cassette.

If you're willing to give this adjustment a try, but you're not sure just how to accomplish the job, do yourself a favor and take a trip down to your favorite HI-FI or TV repair shop. Ask the technicians to show you how to do it. Even if they charge for the service, it is worth ten times more. You'll be amazed at how simple it is and how much easier life becomes when you don't have to hassle with tapes that fail to load.

GETTING STARTED

OK. Pro/File 2068 is now loaded into the computer. The message (also termed "prompt") staring you in the face says to Press C-to CREATE a new file or L-to LOAD an existing one. Early in the introduction Pro/File 2068 was termed the functional equivalent of an electronic file card holder. Right now, your holder is empty.

With Pro/File 2068 it is possible to have many different programs which hold different kinds of information. Procedures in the program allow you to save onto tape your current data base and load something else in its place. You could have one program that stores phone numbers and another that lists your household inventory.

When you see the "L-to LOAD, C-to CREATE" prompt, the computer is asking you if you want to load a data base already started, or create a completely new one. If you have never used Pro/File 2068 before, the only option you have is to CREATE because you don't have any others yet.

At this instruction, and everywhere else you are given the chance to input an option or function, use CAPITAL letters. The program is written to read only capitals as your response. You may wish to put the computer in CAPS LOCK mode at this time to avoid making any mistakes. To do this press "CAPS SHIFT" and "2" to lock on to capital letters. If you want to use lower case letters you certainly can, but be sure to shift to capitals when you respond to prompts.

Anyway, press the capital letter "C" and ENTER. When you do, the computer places another prompt on the screen. It says:

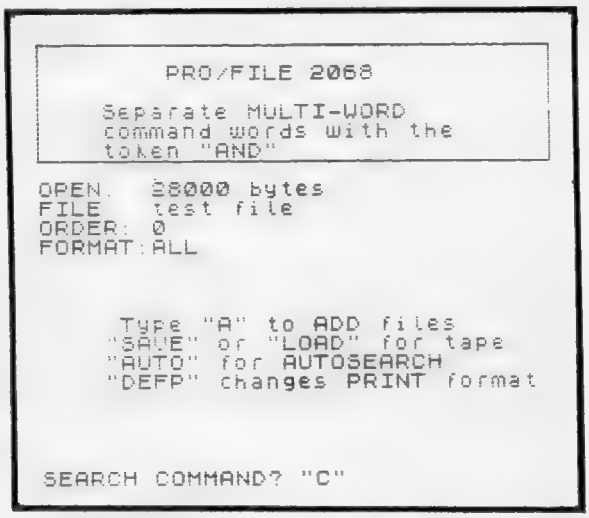
ENTER A NAME FOR THIS FILE

This name is like a label you stick on the cover of your index card box which holds your telephone numbers. A name describes the type of information you intend to store in the program. Later, when you choose the "L-to LOAD" option to load an existing file (or data base) you use the name you give it now to tell the computer which file you want to load.

Choose a name that reminds you of the type of information you plan to add. Make it serve like the title of a book. JUNE SALES, or PHONE LIST, or ADDRESSES would all work just fine. Names can be a maximum of ten characters in length. Type in the name you want and press ENTER. The TV screen clears and the computer brings you to the Main Pro/File Menu, one with which you shall become very familiar.

THE MAIN MENU

This display is like "home base" in Pro/File 2068. It is from here that you start adding records or searching for them. The Main Menu tells you a lot about what to do and what state the computer is in. The menu will look something like this:



Within the title block are the instructions:

SEPARATE MULTI WORD COMANDS WITH
THE TOKEN "AND"

Once you have added a few records to Pro/File 2068, it is from this menu that you enter search commands. You could enter something like SMITH to start the computer searching for each record that has SMITH in it. A "Multi-Word Command" is one where you want to find SMITH's born in JUNE. This promt tells you how to enter this kind of search command. First you type the first word (SMITH). Then you type the token AND (press "SYMBOL-SHIFT" and the letter "Y"). Finally you type the second word (JUNE). More about this later.

Below this are four lines which tell you the condition of your present data base.

OPEN tells you how much space remains open for adding new information. You start with 28,000 bytes or characters. As you add data, the space open decreases. Keep your eye on this number. If you try to add more files than there is space open, the computer will go berserk! DO NOT EXCEED THE SPACE AVAILABLE LISTED IN "OPEN"!

FILE is the name or title of the data base you just created.

ORDER represents the line number of a particular group of records which forms the basis of alphabetized or numerically ordered file displays when using the AUTOSEARCH option.

FORMAT defines the portion of a file thats gets printed. Pro/File 2068 is designed to work with the Timex 2040 printer. The program must be modified if you want to use some other brand of printer. A chapter of this book shows you how to make this alteration.

This portion of the Main Menu gives you access to the current state of the program. At any time you can reference how much space is open for new information, the name or title of the data you are currently working with, and the order and print format.

COMMAND OPTIONS FROM THE MAIN MENU

The next four lines on the Main Menu give five function options. Enter:

"A" to ADD a new file or record to the program.
"SAVE" to allow you to SAVE your records on tape.
"LOAD" to LOAD other existing files from tape.
"AUTO" to produce AUTOmatic printouts, and numerically or alphabetically ordered displays of information.
"DEFP" to DEFINE PRINTER format. With this command you specify which lines of a file get printed.

These commands will each be covered in more detail later. At the bottom of the Main Menu is the phrase "SEARCH COMMAND?" followed by a blinking cursor. After data has been added to the program, you can enter "Search Commands" to access the information from those files. At this point, anything you enter EXCEPT the above five function options is interpreted as a search command.

HOW TO ADD DATA

With the Main Menu on the screen, type "A", then ENTER. The TV screen will clear and at the top left you'll find a blinking ADD/EDIT cursor. The computer is now waiting for you to start entering data. At the bottom of the screen is a box listing various cursor controls and editing functions. You should think of the computer in the ADD/EDIT mode as being just like a typewriter with the TV screen as the sheet of paper you're typing on. The blinking cursor tells you where the next key you press will be printed.

For practice, type in a name such as STEPHEN PAGE. Press ENTER. Touching the ENTER key moves the cursor to the next line. Type in 7 SUMMER STREET. Press ENTER again. Type ROCKPORT, MAINE and ENTER. Type 04856. ENTER. Type 207 236 3659. ENTER. Now you have five lines of information and the beginnings of a file. With this practice file, or any other, try some of the editing commands.

The arrows (shifted 5, 6, 7, 8) move the cursor at will. You can position the ADD/EDIT cursor anywhere on the screen using the arrows and start typing from a new position.

Press SHIFT and "1" to alternate between INSERT mode and OVER mode. When the computer is in INSERT mode, whatever characters you type will be inserted between existing text on the line you're working on. In the OVER mode, the computer replaces (or types over) whatever is on the screen with what you type.

Pro/File 2068 always starts in the OVER mode. You can tell which mode you're in by looking at the color of the ADD/EDIT cursor.

A YELLOW cursor means you're in the OVER mode.
A WHITE cursor means the INSERT mode is activated.

Whenever you change modes by pressing SHIFT and "1", the new mode remains in effect until you change it again. Try putting the computer in INSERT (white cursor) and move the cursor up to the "R" in ROCKPORT. Then type a "B". The "R" moves over one space and ROCKPORT becomes BROCKPORT. To get out of this mode, press SHIFT and 1 again. The cursor changes back to yellow and you return to the OVER mode.

SHIFT 0 is used to delete a character. Move the cursor to the "B" in BROCKPORT and press SHIFT 0. The "B" is deleted and every character on the line is moved to the left.

Now press SHIFT 9 and some more options will be displayed.

SHIFT 2 is the capital letters lock for shifting in or out of CAPS. If you are typing in lower case, the regular CAPS SHIFT still works in the normal fashion, but to lock on to upper case, press SHIFT 2. When you do, you will see the cursor briefly change to a blue color and a ">" symbol will blink in its place. This tells you the computer has gone into upper case. Everything you type will be locked in CAPS. If you press SHIFT 2 again, the "<" symbol is flashed in blue. Now the machine is back in lower case.

SHIFT 3 deletes an entire line from the file you're working on. Every line which appears below the line the cursor is blinking on will move up one position to fill the gap left by the deleted line.

The next command, SHIFT 4, means LINE ERASE. Here a line can be deleted, but lines which follow will not move. The blank space remains. Also, only a portion of the line will be blanked out, that being everything to the right of the cursor. Text on the same line, but appearing to the cursor's left will remain. Therefore, an entire line could be erased if the cursor is in the left most column, but if it is somewhere in the middle of the line, only a part of it is blanked out.

Simultaneously pressing the SHIFT and SYMBOL SHIFT keys causes the computer to INSERT a line. This is just the opposite of the line delete function. Every line from the ADD/EDIT cursor and below is shifted down one space and a blank line is inserted on which you can type new information. It is important to note that if the bottom line has any text written on it, executing the LINE INSERT command will shift everything down and what was on the last line is lost. It cannot be recovered.

SHIFT 9, as before, is used to change the ADD/EDIT menu. When you press it this time, the current status of the program (OPEN, FILE, FORMAT, etc.) is displayed. Press SHIFT 9 again and return to the first set of commands.

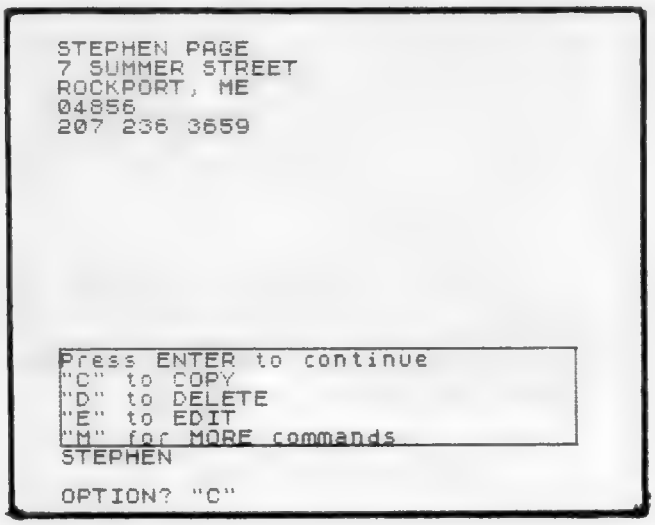
There are fifteen lines available for adding information to the file. Additions and changes can be made anytime the file is on the screen.

To CLOSE the file and add it to the memory, press STOP (SYMBOL SHIFT and "A"). The file is stored and the Main Menu returns to the screen. Note that the number of bytes OPEN has decreased by the amount used by the file you just added.

Continue to ADD more files this way and use the commands to edit them. Note that any edit command will work while you're in the ADD/EDIT mode even if it is not currently displayed in the ADD/EDIT menu.

THE SEARCH COMMAND--YOUR ACCESS TO FILES

Once you have some files in memory, you can access them by typing in any word, group of letters, or group of numbers that you know is in the record you want to look up. For example, with the practice file noted previously, the search command could be STEPHEN, PAGE, ROCKPORT, 04856, 207, STEPH, or any combination or group of characters. With the Main Menu on the screen, type in a search command that you wish. Press ENTER and every file that contains that group of characters will appear on the screen. The computer displays the first matching file it finds and prints at the bottom of the screen a box listing what are called the "Display Options".



THE DISPLAY OPTION MENU

After a file has been retrieved and printed on the screen, there are several things you can do. These options are executed by pressing one of several keys. These are the options you have:

Press ENTER to search through more files and find other matches to the Search Command. If, for example, you use Pro/File 2068 as an address list and you have several people named PAGE listed, you could type PAGE as a Search Command and the computer will find and print the first file it finds that has the word PAGE in it. The machine has no way of knowing if the file it finds is indeed the one you want. Perhaps it is the second or third PAGE you want to look up. If you press just ENTER of the Display Options, the program picks up where it left off and CONTINUES searching for more files that hold the word PAGE. You can press ENTER after every file display to "step" through every listing which matches with the Search Command you input. Searching continues through every file until all your data has been checked. Then the computer prints, "SEARCH IS COMPLETE." This tells you there is no more data to check and some other Display Option should be selected.

"C" chosen as an option will COPY the file printed on the TV onto a printer if you have one attached. The Print Format as defined by DEFP controls just which lines actually get printed. More on DEFP later.

"D" will DELETE the file currently displayed from memory. Once done, the computer returns to the Main Menu.

"E" shifts into the EDIT mode. The ADD/EDIT menu will be displayed at the bottom of the screen and the flashing cursor is activated. The edit procedure is identical to that of adding a new file only the EDIT mode starts with the currently displayed file already printed on the screen.

"M" prints four MORE Display Options. When you press "M" you see these:

"R" will start the search again from the beginning, looking for the Search Command displayed. If after looking through your mail list of 6 different people named PAGE, you decide it was the first one you really wanted, you could press "R" to RETURN to the first.

"A" puts the ADD mode ready to take a new file. This is the same as if you type "A" from the Main Menu.

From the Display Options you can also enter most "Main Menu" commands. A new SEARCH COMMAND of any kind can be typed in at this time to look for another file.

SAVE, LOAD, and DEFP can be typed in to update your cassette, load in another data base, or alter Print Format. (AUTO must be started from the Main Menu).

"M" typed a second time will display the "status" of the program as in the Main Menu: OPEN, FILE, ORDER, FORMAT, etc. The Display Options are printed to remind you of the commands you can use. Even though there are three separate displays, it is not necessary to list the display which shows the option you wish to execute. You could, for example, press "A" to ADD when the first series of options (which does not list the "A" command) is shown. Every option is open to you regardless of whether it is printed on the screen or not.

MULTI-WORD SEARCHES

Sometimes you will want to use two or more variables in locating a file. For example, find all people named PAGE who live in ROCKPORT. This kind of search is called a Multi-Word Search. PRO/FILE 2068 can find files based on two or even more separate and distinct words. Only those files that contain EVERY word will be displayed. For a Multi-Word Search Command, type the first word such as PAGE, the token "AND", then the next word like ROCKPORT.

To type the token "AND", press both the SYMBOL SHIFT and the "Y" key at the same time. Do not spell out the letters A, N, D. It is not the same thing. The program accepts up to 32 characters in a Search Command. As long as you're within this limit, you can hunt for as many different words as you like.

There is one situation you should watch out for and this can best be illustrated by taking another look at the practice file:

```
STEPHEN PAGE
7 SUMMER STREET
ROCKPORT, ME
04856
207 236 3659
```

Search Commands like "PAGE AND ME", or "STEPHEN AND 04856 AND ROCKPORT" will put this practice file on the TV screen, but "PAGE AND 3659" will not. In the case of Multi-Word searches, when the last word of the search command is also found in the last word of a file, the computer will not display the file. On the other hand, a search for "3659 AND PAGE" will find it. As you add data, it is important to remember this or the program may produce some unexpected results. Solutions which insure that this event will never occur include:

-Before closing a newly added or edited file, add a special character like a period or underline (SYMBOL SHIFT-0) to the end of each file. Then, as long as you don't use this character as a Search Command, the computer won't lie to you when you tell it to give you a file display.

-Another alternative centers around the manner in which you set up your records. If you always put the data you'll use to search by at the beginning of a file, and keep other info at the end, this inaccuracy will never come back to haunt you.

Finally, if you use the end of a record to signify a special meaning or category and every record uses this same position to store this same type of information, you can use the data held there in your search command, but enter it first when you issue a Multi-Word Search.

All in all, this unusual situation will probably never occur in your data base. If it does, you now know what is going on and how to fix it. You should not let this (choke) "bug" deter you from using the Multi-Word search. The potential power of a 5 or 6 word Search Command is enormous.

PRINT FORMAT and DEFP

If you use a printer with Pro/File 2068, anything that can be searched for can be printed on paper as well as on the TV screen. But when you use a printer, you can control which lines of an individual file will be printed. The "Print Format" is the way you arrange file lines on paper. It can be changed by typing "DEFP" either from the Main Menu or the Display Options. "DEFP" means "define printer". Typing in this command will give you a prompt which asks you to:

ENTER DESIRED FORMAT

It is here that you tell the computer which lines of a file you wish to lprint when you decide to print something out on paper.

To print the entire file, type "ALL". To print just part of a file, list the line numbers you want to print and separate each with a "/". Any combination will work. You can tell the printer to print blank lines by typing "0" in place of a line number. For example, the format 1/2/3/0/6/0/5 would cause lines 1, 2, and 3; followed by a blank line; then line 6; another blank line, and finally line 5 to be printed out when you press "C" of the Display Options to "copy" a file display on paper.

Once you define the PRINT FORMAT, it will be displayed on the "status" chart of the Main Menu for easy reference. A status report is also available from the Display Options and the Add/Edit Menu. After PRINT FORMAT is set, it remains in effect until you change it by issuing the DEFP command again.

AUTOSEARCH

Automatic searching allows you to 1) print out more than one file at a time; and 2) arrange files so they are displayed or printed in alphabetical or numerical order. Therefore it is possible to alphabetize an index or address list, order zip codes numerically for bulk mailing, or countless other uses.

To use this function, type AUTO from the Main Menu. The display asks: PRINT OUT? (Y/N). Enter "Y" if you want all files to go to the printer. Enter "N" or any key except "Y" to put the data on the TV screen only.

Next type in a Search Command when the computer asks for one. If you do not type a Search Command and instead type just ENTER, the computer leaves the AUTO mode and returns to the Main Menu.

Now you're asked to type in a line number. This will be the basis for ordering. For example, if you add files such that the fifth line always holds a zip code, and you want the files sorted by zip, you would type in "5". If you want an alphabetical ordering of names which are always listed on the first line, you would type in number 1.

You can also tell the computer that you don't want the files ordered. When the program asks for a line number, type in a "0". You'll then get a readout of all the files containing the Search command as they are found rather than in any particular order.

After you type in a line number, press ENTER, and the computer will scan all your records automatically, printing the file that comes first in the sequence you have chosen. If you choose printer output, the ordering progresses until all files are printed. When TV display only is selected, each file is displayed in turn with the Display Options. Continue to press ENTER to order all the specified files.

THE ASTERISK--A SPECIAL SEARCH COMMAND

Although you never see it, Pro/File 2068 marks the start of every file with the asterisk (*). Because of this, the "*" becomes a useful tool for listing all files in memory. Type in a "*" as a Search Command and every file will be displayed or printed.

You can also use the asterisk while adding and editing files to fool the computer into thinking there are two separate files on the screen. When a file becomes large and you feel it might be better to "split" it up into two different ones, insert a "*" at the appropriate place. The next time you look the file up, you'll find that what once was one file, now is two.

SAVING YOUR DATA BASE

As you create files, add to them and edit them, you should save the files on tape often. Every time the computer is shut off the new material in memory is lost. Fortunately, it is easy to SAVE your programs and files. Use a new tape in the recorder and start it recording. Then type SAVE from the Main Menu and press ENTER. Be sure the cables are connected between the recorder and computer. The normal SAVE pattern will appear on the screen, and when SAVING is complete, you will return to the Main Menu.

LOADING YOUR SAVED FILES

When you are newly loading Pro/File 2068, you are given the choice to press "C" to CREATE a new data base, or "L" to load an existing one. This is one time that you can load the files you have stored on tape. Otherwise, type LOAD from the Main Menu. You will be asked to type in the name of the file you want. Enter this name, and press PLAY on the recorder. The data base you choose will be loaded into the computer. It will NOT load anything which has a name different from what you type in.

When the computer asks you to type the name of the data base you wish to load, you can also press just ENTER. The effect here is that the computer will load the first program it encounters on the tape regardless of the name it was saved under. This is handy if you have forgotten the name of the tape you're trying to load. If the program is one of your previously saved Pro/File's, the new data base will be loaded. It could also be some other Basic program which you use. The computer really doesn't care what you try to load so pay attention to what you're doing when you use just ENTER in place of a name when you load. The flexibility is nice, but it requires a little extra care to insure you don't load the wrong program.

You will notice that a previously saved data base takes longer to load than the original Pro/File 2068 cassette. This is normal. The reason why this is so is that the original tape does not have any data in it. Consequently, it requires much less time to load. Copies that you have saved yourself, contain 28,000 more bytes of data than the original.

It is important to remember that in order to load one of your previously saved data bases, you must start by loading the master tape of Pro/File 2068 first so that Pro/File's machine code is present in the computer. You don't need to load the master tape every time you want to load your own tape--just once before you start is all that's required. This means that when you're ready to start a session with Pro/File 2068, load the master. Then, as long as you don't turn the computer off, you can load any tape which you have made any time you wish. You can flip back and forth from one data base to another. You can even load completely different programs and then go back to one of your Pro/Files. But if you turn the computer off and thereby erase Pro/File 2068's machine code, you must first load the master tape before you load one of your previously saved data bases. If you try to just load one of your own tapes without first loading the master, the computer will crash because it will try to execute a machine code program which was not loaded first. The results are not always predictable, but generally, the screen goes black. After a few seconds some lines grow up from the bottom of the TV screen. Whenever this happens, turn off the computer and start over. Pay particular attention to this paragraph.

Don't be concerned that the master tape will someday go bad and you will no longer be able to use any of your files. Later in this book are instructions on how to make back-ups of the master tape so if it ever does go bad, you'll have another copy.

APPLICATIONS
How To Make Pro/File 2068 Work For You

Now that you have an idea of how to put information into Pro/File and also how to search, find, and display that information when you want it back again, you need to ask yourself:

A) What exactly do I want to store in Pro/File 2068?

B) How can I set up my files to make the most of the program's file finding capabilities?

C) How can I file the largest amount of data in the least amount of memory space?

The answers to these questions are different for every individual. Pro/File was designed for speed, versatility, and capacity, but without at least a little forethought on how to best enter your data, you might miss out on some of the program's capabilities. Here are some examples that might give you some ideas.

Customer/Client Record

Suppose you are a sales person and you rely heavily on the telephone for communications with your clients and suppliers. What you need is fast access to telephone numbers. Also, a brief history of past dealings with each customer would be nice. Some of your clients require regular periodic service and you need to know WHEN to call as well as just WHO to call. Finally, if a customer calls you with a problem or request, you need to know where to find all pertinent information about the customer and the situation.

A phone directory of just names and numbers would be an elementary task for Pro/File 2068. You could store your names and numbers as numerous one line files which contain both the name and the number. If you could keep each file limited to a maximum of one line, you could store something over 875 names and numbers.

Any time you want, you could make a search for the name of the person you wish to call. Type a person's name or a portion of it--every file that the name is found in will be printed one after another.

If you arrange your names and numbers so a person's last name comes first, you could produce an alphabetized listing of everybody in the program by using the asterisk (*) as a search command.

If your customers buy several products from you, it might be useful to include information on just what they buy along with their name and phone number. You could add more lines to each file. In great detail you could

insert the date of purchase, item, amount, and whatever else you need. Remember, however, that as you add more data, you slowly deplete the memory space available to the computer. There are ways to maximize this limited space by devising codes for certain entries.

Coded Entries Save Memory Space

If you sell 10 different products and you need to record who buys what, and when they buy it, you could enter your clients' phone directory as usual. Then for each client, you could assign a date. With the date could be codes which represent your customers' purchases.

Pro/File 2068's search mechanism does not distinguish between a coded entry like "C"--meaning "CATALOG REQUEST" or the letter "C" in a file called "CHURCH DIRECTORY". If you input "C" as a search command, every file with a "C" in it--regardless of meaning--will be displayed.

Coded entries (individual letters or characters that are intended to mean something else) can be used along with another character (a code separator) like a colon or dash to have special meaning. If you use these "separators" only with coded characters, you can use them as a search command to find just the information you need.

For example, you could insert a "C-" in a customer file to mean that this person requested a catalog. If you want to find all those people who asked for a catalog, you could enter "C-" as a search command to find them. Such a code uses just 2 bytes of memory while "CATALOG REQUEST" uses 15. After you have a hundred people requesting catalogs, the memory savings adds up to 1300 bytes, and that ain't hay!

As long as you stick to one method of inputting various code letters, you can make searches for the code with the separator and never risk pulling up some unrelated file that just happens to have the same code letter in it. Here is what a typical "customer file" might look like:

```
DPUb
John H. Jones
P.O. Box 1325
Mobile, AL 36609
4/28-C-9.95-DPU-
5/15-›-4.95-DPUb-
```

Lines 5-6 each hold 4 pieces of information about Mr. Jones. Each code is separated by a dash. First is the date of the transaction; second, the origin of the order; third, the amount paid; and fourth, the item ordered.

The breakdown for each line is:

Line	Code	What it means
1	DPUb	A product code meaning: "Doggy Picker-upper baggies"
2-4		Customer's name and address
5	4/28-	Date of transaction
5	C-	Signifies response to an ad in "Creatures Magazine"
5	9.95-	Amount of transaction
5	DPU-	Item purchased (Doggy Picker Upper)
6	5/15-	Date of second transaction
6	>-	Signifies repeat order
6	4.95	Second tranaction amount
6	DPUb	Item sold (replacement baggies)

A great deal of information about a business's sales can be gleaned from files storing data in this manner. Mailing labels of any particular group of customers could also be produced.

If your entire customer list were set up in this fashion, you could initiate search commands like:

Command	To tell you
DPUb-	All sales of "baggies"
4/28-	All transactions on April 28th
C-	All respondants to "Creatures Magazine"
>-	All repeat orders
DPU- AND C-	All customers who purchased "Picker-uppers" from ad in "Creatures
Jones	All customers named Jones
4/	All transactions in April
DPU- AND DPUb-	All who purchased both "Picker-uppers" and replacement "baggies"

Any search can be displayed on the TV screen or sent to the printer. If printer parameters (DEFP) are set so that only the first four lines of each file are printed out, you can put your customers' names on labels. When you actually get down to the business of packaging shipments, the product code on the top line tells you what to send the customer whose name appears on the label.

Remember that the asterisk (*) can be used to step through every individual file. If you want to send out a bulk mailing to the names in your list, you would first set printer parameters so that just lines 2-4 would be printed. Then you would perform an AUTOSEARCH using the asterisk as a search command. To produce the list in zip-code order which is a requirement of bulk mailing, you would need to reserve one line of each file to hold the customer's zip. Then you could order the print out on the basis of that line.

What You Ask For is What You Get

Whatever codes you decide to use, be careful to insure that they will appear in files ONLY as codes because if they're not you could get some unexpected files as a result of a search. This applies to non-coded search commands as well.

Once when I was showing off the program to a person who was skeptical about the usefulness of a personal computer, I was showing how easy it was to pull every person from Alabama. I typed in AL for a search command. Much to my chagrin, the first address to come up was ALUMINUM PRODUCTS, INC. in New York. The second address was just as bad--ALTERNATIVE TECHNOLOGIES in California. I continued searching for Alabama. Only three of eight addresses actually actually listed Alabama as the state. The computer made no mistakes. It correctly displayed every file that had AL in it, but AL may not necessarily appear only as a state.

To insure that only Alabama is found, look at the Jones customer file again. Note that both before and after the abbreviation AL (meaning Alabama) there is a blank space to separate it from the city and zip-code. If you include these spaces in your search command, you'll eliminate all the unrelated garbage that the computer finds which also matches with "AL". In other words don't search for just "AL" to find Alabama. Search for "(space)AL(space)" instead.

How To Remember Your Codes

At first it may seem that using many different codes to mean different things will be very confusing. Actually, the hardest part is deciding what codes to use. The more you use them, the more second nature they become. One way to help you remember what your codes are is to create a HELP FILE. All you have to do is Add a file. On the first line type "HELP!" Then as you add new codes, add them along with their meanings to the file. Make your explanations as friendly or as abusive as you wish. The next time you forget a code, just type "HELP!" as a search command. This is not only useful to jog your own memory, but if others use your data base, they will appreciate your thoughtfulness at deciphering all your cryptic codes.

PRO/FILE 2068 the Information Directory

Until we TS2068 owners figure out how to make use of the computer's memory banking capabilities, we will have to restrict our data capacity to the rated 28,000 characters. This is not enough for really big jobs. One way you can work around this limitation is to use the program as a catalog file rather than a holder of massive quantities of data.

Instead of using Pro/File 2068 to hold information, let it tell you where to find it. Let's say you have a room full of files, records and general clutter. Records could be stored under any number of categories. You need some way to index all your files so that once you decide what you're looking for, you can figure out where to find it. Here comes Pro/File 2068 to the rescue.

Every time you have a piece of literature, a receipt, or a client record that needs filing, log it into the computer. Along with the name and description of the information, include the date filed and what drawer, closet, or shoe box you stuffed it in.

When you need that piece of information back, you could search through the computer under name, date, or type of data to find the drawer to look in. Similarly, when you're cleaning out your files and you come to box number 47, you might ask "What IS all this stuff?" Pro/File 2068 could tell you.

PRO/FILE 2068 the Sales Account Schedule

Traveling sales people, manufacturers representatives and others, whose job it is to contact various accounts, sometimes have problems with scheduling, business locations and missed appointments.

Such difficulties can be minimized with a file containing not only pertinent data but also a graphic reminder. In this instance, FORMATTING your file displays is important for two reasons.

The first is to be certain that similar information is always entered on the same line. In this way the printout can be formatted to generate reports based upon specific data (products purchased, next visit date or city).

The second reason is that if your filing system is well organized, chances are that you, too, will become better organized.

What should go into this type of file? An account identification should be on the first line. A number or name usually is given to the accounts of most companies. Consistency with your company is important, so use an established method in this case.

The name of the company you contact, the person you see, an address and phone number are necessary as well. Filing this data will help not only you but also a substitute should you not be able to make the next trip.

If your product line is large or varies greatly but appeals to specific needs of various accounts, Then a "products purchased" line (or "field") can be of great help in managing this type of data base.

For instance, let's assume that your product line consists of spoons, shoe laces, computers and how-to books. With those items it is probable that very few (if any) of your accounts have an interest in all of those products. So including the products purchased with the ability to search for that data can organize your planning. And complementing that data with a "next visit" date can help in planning the itinerary of your next sales trip.

Most account data bases go just this far with data filed. There is an additional capability of Pro/File that can be put to very good use. Assume that you have a large number of customers that must be visited and satisfied. To this responsibility, your boss adds new accounts every month. Remembering the physical location of your most recent accounts is difficult enough.

Pro/File 2068 can help you find locations. How? Just include a map as part of your file. To do this you can use the symbols accessed by the "symbol shift" key. Use the underline for major streets, the colon for side streets and other symbols for landmarks. Placing a "#" and company name on your map will pinpoint account locations. Study the example and experiment.

Sample Display Format for a Sales Account Scheduler

```
ACCOUNT: x02068
NAME: ABC CORP.
CONTACT: CLIVE
ADDRESS: 12 MAIN ST
         COMPUTERVILLE
PHONE: 555-2068
PRODUCTS: COMPUTERS, SPOONS
MAP:
      U:                M:
      I: ABC CORP.      A:
   ___N:_______#_____   I:_______
      E:  MAIN          N:

NEXT VISIT: DEC 10, 1984
```

Can you initiate a useable search based upon the maps in your file? A qualified YES is the answer. Usint the example, you could search by the street and city name. The display and/or printout would show all your accounts on MAIN street in a particular city.

With a little imagination you could tailor this file to your specific needs.

Recipes

It's been a difficult day at work and going home through rush hour traffic worsened your attitude. You're not up to the hassle of standing in line at a restaurant and you decide to make something quick to eat at home. A search through the cupboard produces only a can of kidney beans. Next you open the refrigerator and see a bottle of ketchup and leftover hamburger. UGH! Looks like a fast food night.

So you brave the traffic (again) and end up with a burger, fries and a shake. Just as you finish your "American cuisine", a friend walks by and tells you about a terrific chili (your favorite) recipe using kidney beans, ketchup and some left over hamburger! After you finish shouting "Where were you half an hour ago?" you realize the potential of a recipe data base.

Here is one way you can start to assemble and format such a file. On your next few trips to the supermarket, pick up a few of those magazines at the checkout counter. They usually contain a few quickie recipes dedicated to leftovers.

Once you've collected some concoctions that interest your palate you will be ready to devise a simple but very useful "RECIPE" data base. It might look like this:

```
NAME: CHILI
INGREDIENTS:
1 can kidney beans
1 pinch chili powder
1 Tbsp chopped onion
1 lb hamburger

INST:
Mix together after warming
hamburger. Let simmer for
15-20 min.
```

Limiting your total field to 200 or 220 characters should give you a potential file of 125 recipies.

You can use multiple word search commands based on the ingredients at hand to become a "computer gourmet." You could, for example enter "kidney beans AND hamburger AND ketchup" as a search command. Whatever recipes you have with these ingredients will be found and displayed.

Cooking times can also be used as a search command. If you find yourself asking "What can I fix in 20-30 minutes?" try asking the computer instead.

PRO/FILE 2068 the Resource Utilization Data Base

This application is for all those who did not read the last application about recipes. You can take the idea of a recipe and apply it to any situation where you have a project to be constructed, formulated or cooked, and you wish to know what can be done with the resources at hand. An actual example that comes to mind is the "What in the world are we going to do today?" dilemma faced by teachers, scout leaders, or any other person who finds himself in the position of finding enlightening interesting projects for others to do.

As you discover or invent "recipes" for unusual projects using discarded, recycled inexpensive materials, put them into Pro/File the same way that a food recipe would be entered. Here's a "recipe" for a play telephone that kids just love:

```
PROJECT: KITE STRING TELEPHONE
MATERIALS:
20 feet of kite string(+ or -)
2 paper cups

INST:
Punch holes in cup bottoms using
tooth pick or paper clip. Push
ends of string thru holes. Knot
the ends to secure.

Pull string taught. Speak into
one cup and listen at the other
end.
```

Now when someone drops off 4 dozen paper cups and a mile of kite string you will never go wanting for a project for your kindergarden class or den of cub scouts.

PRO/FILE 2068 for Model Railroaders

Model railroading is a hobby that is great fun. Pro/File 2068 can provide help in two important areas of this activity.

The first is as a roster (catalog) of locomotives, freight and passenger cars. If you are new to the hobby and your collection is still rather small, proper formatting might not seem to be very important. WRONG! Do it right the first time and save yourself a lot of trouble later. Format your file so that retrieval will be both easy and useful.

The first line should contain the type of locomotive or rolling stock. In that way you could enter your purchases as they are made. Then, when you want a list of all your locomotives, a search for "LOCO" will provide that particular roster.

Including a line for "DESCRIPTION" can save a great deal of time should you want the following items:

Loco AND Steam
Passenger AND Pennsylvania
Freight AND Gondola
Loco AND MofW (maintenance of way)
Freight AND Reef AND UP (Union Pacific)

If you want to include reporting marks (further identification) that data could be included as well in order to more accurately search your files.

Purchase price and value for each piece are important too. Should something be destroyed, lost or damaged, you can use the data to make an assessment. It can also be used when you find the need to sell some of your equipment.

Having an "OTHER" line may seem unnecessary at first. Here are some suggestions for helpful data: Use that field for text to record 1) a history of the item, 2) other types of equipment--loco, caboose, etc.--that the item can be used with, 3) when a particular item is due for repair or maintenance, 4) if an Item is to be sold or traded, 5) what state the item is in--kit, assembled, or junk.

Here is a sample format for a model railroad data base:

```
TYPE: LOCO, FREIGHT, PASSENGER
DESCRIPTION: Class, Type, Size
             Capacity
PURCHASE PRICE:
VALUE:
OTHER: Kit, Assembled, Source
       Material
MARKS:
LAYOUT LOCATION: XYZ Company
NEXT LOCATION: ABC Company
```

When your roster has been completed, make a copy as a master file to be updated when you acquire new items.

Why has space been allocated for "LAYOUT LOCATION" and "NEXT LOCATION"? That is the second important use for a model railroading file, because that data will allow Pro/File to become actively involved with the operation of your model railroad. Here's how it works:

First load your master file and then put the cassette in a safe place. After your last operating session, you should know where specific cars are located and that the requirements of all industries along the way have been satisfied. Enter a search command for a car type/description/and reporting marks (if any).

When the file appears, call for the EDIT mode and update its layout location. When all cars on the layout have been identified with a location and filed, make a copy on a cassette and label it "RAILROAD OPERATION".

When it's time to determine the activity for the next operating session, load the "OPERATION" tape and update the "NEW LOCATION" line. Copy the file of each car going to a new destination and you're ready for another session.

Start with a few cars to be moved until you become accustomed to working with your new "dispatcher." With a bit of practice, Pro/File can become an active participant at your railroad operating sessions.

PRO/FILE 2068 the Foreign Language Tutor

You can use Pro/File as a teaching aid for foreign languages. The way to do it is to create many two line files that use one line to hold first, the initial of the language used and second, an English word to be translated. On line two type in the foreign translation. Files would look like:

```
G GOOD                F GOOD
GUT                   BIEN

F LISTEN              S GOOD
ECOUTER               BUENO
```

When you are ready to use your language data base, enter a search command like F GOOD, but before you press ENTER, try to guess the proper translation. After you decide the right word, hit the ENTER button and let Pro/File pull up the English word and its foreign equivalent.

You are not restricted to just one language. As long as you place the initial of the language you want in both the search command and the file itself, you will get translations only in the language you specify. Searches for F GOOD, I GOOD, S GOOD, or R GOOD would all result in different translations (provided you entered them into your data base first).

PRO/FILE 2068 the Ready Reference Identifier

The language translator last described showed in very specific terms how to create an "associative" data base--a collection of word pairs which mean the same thing. When you search for one word, you find the translation "associated" with it. For Pro/File, translations and associations are a piece of cake. You can expand the example of foreign words to any number of applications.

Imagine a thesaurus or data base of synonyms. Instead of holding an English word and its French equivalent, you could have a word and its synonyms.

Associating tables and charts with a word you use to search with is also very useful. If you're an electronics buff like me, you love to try to make electronic circuits work even if you don't know much about the theory involved. I was forever pulling out references that hold charts of resistor color codes and equations of ohms law until I thought to put all that data into Pro/File 2068. I have a special "Electronics" cassette which I load before I start wiring up my latest project.

My data base has a color code chart like the one below which I call up by typing "COLOR" as a search command.

```
RESISTOR COLOR CODES
Color    Band 1  2  3(Multiplier)
---------------------------------
BLACK          0  0  1
BROWN          1  1  10
RED            2  2  100
ORANGE         3  3  1000
YELLOW         4  4  10000
GREEN          5  5  100000
BLUE           6  6  1000000
VIOLET         7  7  10000000
GRAY           8  8  100000000
WHITE          9  9   (none)
```

Other data I find useful is IC pinout specs for different chips. When I want to find out which pins go where for some chip I'm using, I just search for the chip number and there it is.

```
74LS00 7400 QUAD NAND GATE
Pin      Function
--------------------------------
1,2      Gate 1 Inputs
 3       Gate 1 Output
4,5      Gate 2 Inputs
 6       Gate 2 Output
 7       Ground
 8       Gate 3 Output
9,10     Gate 3 Inputs
 11      Gate 4 Output
12,13    Gate 4 Inputs
 14      Vcc +5VDC
outputs go lo when both inputs
go high.
```

Pro/File's associative powers can also be used to identify unknown items. A data base used to hold descriptions of such things as plants, animals, replacement parts, world records, or your store's inventory could be searched on the basis of identifying characteristics to find the identity. Here are a few examples.

Let's suppose you have a data base listing different street drugs and their pharmacological effects. You work as a volunteer on a crisis intervention hot line and a person calls in a panic stating that they found a friend who appears to be suffering from an overdose of something. You need to find out very quickly what drug or poison might have been taken.

It is very possible that the person on the other end of the telephone can tell you quite a bit about the condition of the stricken individual. By using clues like YELLOW CAPSULES AND NAUSEA AND DILATED PUPILS AND HALLUCINATIONS as a search command you might be able to make an educated guess as to what drug was ingested. If you can give this kind of information to the rescuers, it might just save a life.

In another instance, say you are an amateur botanist. You have compiled a large description of native wild plants. Files are set up so that identifying characteristics accompany the botannical and common names of plants that grow in your area.

When you go hunting for plants you note the features of the unidentified species you find. Then to determine their names you type into Pro/File 2068 a search command fitting the description of the plant in question. Such a command could be "<12in. AND square stems AND opposite leaves AND toothed leaves".

If your "plant" data base was set up as follows, such a search would find this file--SPEARMINT.

```
MENTHA VIRIDIS or M. spicata
Labiatae family
Spearmint, Green Mint
DESCRIPTION:square stems,
<12in., creeping roots,
opposite leaves, toothed leaves
smooth surfaces, prominent ribs

Leaves and stems emit a charac-
teristic minty odor when crushed.

Native of Mediterranean. Once
used medicinally to calm upset
stomachs. See other species of
MENTHA and Labiatae plants.
```

In business, the "associative Pro/File" could be put to good use cross referencing stock numbers with product descriptions. For example, if you sell a large variety of fasteners ranging from staples to molly screws, you could put them in Pro/File with their stock number, shelf location, price, as well as the name of your supplier. One file might look like this:

```
Lag Screw Stock 4725A
5/8x3-1/2
.25 each 10/$2.00
Reorder from:Dick Cunniff
                (512) 334-2423
Bin 325
```

If one of your customers wants 25 5/8 dia. lags but he's not sure of the right length you could enter a search like "Lag Screw AND 5/8x" to produce a list of every length of lag screw you have in stock.

Similarly, if one of your larger customers sends you a purchase order for your catalog number 4725A, you can search for the stock number to find out which bin they are located in.

When re-ordering time comes round, and you notice that bin number 325 is running low, you can search for "Bin 325" to produce the name and number of your supplier.

PRO/FILE 2068 the Travel File

It's been a long year, but finally your vacation has arrived! Now another problem has popped up. Where can you go and how soon can you get appropriate information? Places to go, things to see and where to stay have suddenly beome your primary considerations!

Making use of Pro/File 2068 could help alleviate problems and might even make vacation planning fun. The trick to making a "VACATION" file useful is to collect and enter information throughout the year(s). Proper formatting will make retrieval fun for the whole family. Look at this sample:

```
AREA: STATE, COUNTRY, REGION

ACTIVITY: HIKING, SIGHTSEEING
          SWIMMING, HISTORY
          BEACH, PARK

TRAVEL TIME:      DISTANCE:

LODGING: NAME
         ADDRESS
         PHONE
         RATES
```

A file like this can be search in quite a few different ways. If previous vacations have developed into the same activity year after year, try searching for historic tours, sightseeing or by some other activity you have placed in the data base.

Perhaps the time you have to enjoy is limited and you don't want to waste it traveling. Including the approximate travel time to a specific location will help you find the optimum vacation spot. Distance data can be used for the same reason.

If money is to be the determining factor, searching for motels or hotels with a desired rate can be quite helpful.

Does this eliminate the need for brochures and other similar advertising material? Not really, because those items can be used to expand upon a vacation planning activity. Since Pro/File can be used to set the parameters of your vacation, the confusion that arises when you look through a pile of brochures should be minimized because you will be interested only in locations and activities found by the defined search.

PRO/FILE 2068 the Magazine Article or
Abstracts Indexer

Many of us who use Timex computers have amassed a collection of magazines, newsletters and general information from the print media. Naturally, some of the information we have is valuable and some should be forgotten.

This situation applies to other areas of our lives as well. We receive information related to our special interests, hobbies, professional and general interests. Remembering the truly important facts is difficult enough, and trying to locate a source of information at some later date is sometimes next to impossible!

You can minimize and possibly eliminate fustration by creating a data base that contains lines devoted to subject, source (periodical name and issue date), and article highlights. Such highlights could be "key words" taken from the article itself which give an overview of the subject matter covered in the article.

See if this format will work for you:

```
SUB: UDG(user def graph)

SOURCE: Sync, 1984, Mar-Apr
        pg.40

To define and store 21 char.
3 listings
TS2068
```

This format allows you to search for a specific subject, magazine or a keyword contained in the article highlights.

Every time you run across a piece of literature worth saving put the proper information into Pro/File 2068. The next time you need to refer to that article about user defined graphics, but you can't remember where you saw it, search it out with Pro/File. After the right article is displayed, you need only go to your stack of publications and quickly locate the proper issue.

Besides just helping to find data you need, Pro/File can aid you in your research projects by storing other pertinent information such as references to other works, authors' names, or anything else you want. If you keep your sources in Pro/File for the book or paper you are writing, you can use the program to print out your footnotes and compile a bibliography.

PRO/FILE 2068 the Collector's Dream

Pro/File is a "natural" for organizing collections. Whether you save coins, stamps, model kits, records, or only bills, anything that can be broken down into specific facts can be easily filed and retrieved.

Those facts can differ greatly from one collectible category to another. It is, therefore, extremely important to consider all important variables when you initially format your file.

Two areas of information that we can all relate to are original price and current market value of individual items. This is important not only to be aware of the value of your collection, but also to determine prices if you plan to sell off your collection from time to time. A search based upon current value can quickly indicate items within a specified price range.

Coins and Stamps

While there are numerous ways to organize files for coins and stamps, the following examples will get you started. Modify your file to fit your needs and methods of collecting (domestic, foreign, specific time periods, etc.).

```
COIN:                    CAT:
DESCRIPTION:

YEAR:                  MINT:
CONDITION:
PURCHASE PRICE:
CURRENT VALUE:
GENERAL INFORMATION:
```

Using a "COIN" field will allow you to enter denominations, a specific name or other factor. Any identity that can be used as a search command will do.

If your collection is organized in binders or cases, you might want to include that data under CATalog. Specific coins in a large collection can then be easily be located.

Under GENERAL INFORMATION, data could relate to the previous owner, others who might want that coin, duplicates or a special character that indicates your "want list".

How useful is a "want list" flag? VERY! Include in your file descriptions of coins that you want or need in order to complete a certain area of your collection. At the end of the general description field, place a character (such as +, =, or something that you will never use elsewhere) that will indicate a wanted item. Then, before you go to the next coin show, use that character as a search command. Only those files containing your identifying character will be displayed or printed.

After you have acquired the wanted item, update your file by calling the EDIT mode and remove the special character.

A similar approach can be used for stamp, record, and model kit collections.

Format for Stamps

```
STAMP:                  CAT:
DESCRIPTION:

CHARACTERISTICS:

CONDITION:
PURCHASE PRICE:
CURRENT VALUE:
GENERAL INFORMATION:
```

Format for Model Kits

```
TYPE:car, boat, plane, train
     military, space, bldg.
ITEM:kit name
MANUFACTURER: Company
PHONE NUMBER:
DATE PURCHASED:
ORIGINAL COST:
PRESENT VALUE:
DESCRIP:Date built, Material,
        etc.
```

Format for Records

```
STYLE: EASY,JAZZ,R+B, COUNTRY
       FOLK,CLASSICAL
ARTIST: NAME
TITLE:
SPEED:
SIDE A:          SIDE B:
```

If you collect other objects, studying the arrangement of these samples might help when you organize your files.

PRO/FILE 2068 the Household Inventory

Have you ever looked at your posessions and wondered about how many "things you actually have? If you have the need to itemize and record all your worldly possessions either because of insurance needs or idle curiosity, Pro/File 2068 can be of tremendous help. And while maintaining such a record is justifiably important, be sure to keep at least one copy of that file in a good safe place away from your domicile.

Items for an "inventory" file should consist of furnishings, appliances, cookware, housewares, tools, lawn and garden equipment, clothing, vehicles and computers. Collectibles and/or hobby collections should have their own file since retrieval might be based on special parameters.

Formatting such data for easy retrieval should be your primary consideration. Since we usually think of our possessions in generic terms (i.e. car, table, shirt) the first line of a record should contain the item name. Next you might include a more specific description, serial number, date of purchase, price, value and where you keep the item.

If maintenance is required or repairs have been made, that data could be recorded by date of occurrence. Doing this will give you the capability of entering a search command such as "Maint AND April" which would bring to th display all items requiring Spring maintenance.

Since updating a file is not an everyday occurrence, it might be a good idea to make a FORMAT page for future reference. It is better in this case to refresh your memory rather than go through the file as if it were an "adventure game".

```
ITEM: COMPUTER,FURNITURE
      STEREO,APPLIANCES,ETC.
DESCRIPTION:COLOR,SIZE,TYPE
SERIAL NO.
DATE PURCHASED:
PRICE:
VALUE:
LOCATION: ROOM, OTHER

REQ'D Maint.:
```

One other thing to be aware of is that insurance companies may require different data. Check with your agent before you determine your file format. It is less tedious to include the proper and necessary data the first time.

DON'T FORGET to put a second copy (or printout) in a safe location.

PRO/FILE 2068 the Sporting Events Organizer

Keeping track of the who, when and where data of your favorite sporting teams and events can be complicated if you are involved in organizing sporting events or if you are just an avid fan. Computerizing your scheduling can save a lot of time and Pro/File's ability to retrieve data in various ways can tell you what events will take place on a specific day or within a given month.

When you create a "SPORTS" data base use one line per record to define the sport. This will allow you to search data for specific kinds of sports. Searching by date will automatically limit the display to events on just one day. At this early point in formatting the file you can understand the importance of breaking down the data into as many specific facts as possible.

For instance, if football is to be part of the SPORT field, you can use a "class" field to define leagues. This will enable a search command like NFL, NFC, AFC, NCAA, USFL, CANADIAN, or whatever "class" of football that interests you.

A "date" field is certainly necessary for the information it will provide, but it, too, can be used for searching the data base. Here's how it can work: you've defined the year as 52 individual weeks and you want to know what USFL games will be played in the 21st week. If your search command is USFL AND 21-84", all games scheduled for that week will be displayed.

That data, however, can be found anywhere. To make your data base more useful, include information such as TV station, broadcast time and pertinent comments relating to players, previous records, etc. Using a WINNER field to update the file with scores or results will allow you to build a reference file.

By using Pro/File you can monitor baseball, foot ball, basketball, auto racing, horse racing, tennis, hockey, and the olympics using just one data base. It is important to remember that you must develop and maintain your file as soon as schedules and results are available.

```
SPORT:                    DATE:
CLASS:
EVENT:

TV:
TIME:

COMMENTS:


WINNER:
```

PRO/FILE 2068 the Cost Estimator

Your family size has changed. A computer has invaded your living area. You need more space but the economic situation will not allow a move to a larger house. And because of finances you need to know how much space you can afford if and when you decide to remodel or add on to your home.

A data base pertaining to material and construction costs can be helpful if you want to develop preliminary cost estimates. Architects, engineers and contractors subscribe to cost estimating services that provide specific data, but you need not spend money for similar data.

You, too, can (and probably do) receive such information. That data can be found in your local newspaper, lumber yard flyers or remodeling magazines. But collecting cost data can create a large pile of paper that is usually difficult to put into a useable order. Pro/File can help eliminate that mess.

Construction projects almost always involve the use of a great many different materials whose costs vary with market conditions. It is therefore necessary to develop a file that will allow easy updating of data.

Since many different building materials will be recorded, you may find it useful to make the data base "menu driven". If you first create a record that lists all the materials listed in the data, and you give this record a title such as "MENU", you can then search for "MENU" to call up this record to remind you of what you have listed. This sample should help you get started:

```
MENU...Use lower case to search
MASONRY          ROOFING
LANDSCAPE        GRAVEL
LUMBER           EXCAVATION
WINDOWS          TRIM
FLOORING         TOOLS
CONCRETE
CONTRACTORS
HARDWARE
PLUMBING
FURNITURE
PAINT
DOORS
WALL PAPER
```

Create a new record for each category listed in the "menu" record. Your masonry data fields might look like this:

```
masonry
concrete block
$1.91 per block

brick
regular:$ 89 per 100
special:$120 per 100
glazed: $130 per 100

floor tile
9x9     $150 per 100
12x12   $180 per 100
```

When defining "menu" categories it is a good practice to list them entirely in upper case characters while in each individual category use lower case letters. If you do not use this convention, the menu will appear each time you enter a search command like "masonry" because that "word" is also in the menu. For practice, make up a menu page using the example but use lowercase characters. Then add a few material records also using lower case. Enter some search commands for various headings you have added and watch what happens.

If seeing the menu during each search is tedious or useless, you can correct the menu without destroying your work to this point. Call the "MENU" file and enter the EDIT mode. Overwrite your categories using capitals. After you close the file make a few more test searches using lower case characters. The value of this change will be appreciated as your data base grows.

Once you have a fair amount of pricing information logged in Pro/File you need only search by category to get the prices you want. Remember, however, that materials prices change frequently. Updating old prices with new ones is vital. This is the "garbage in-garbage out" phenomenon working in reverse: Good data on Friday becomes garbage on Saturday if you don't update it.

PRO/FILE 2068 the Geneology File

The genealogical researcher has a lot in common with the historical researcher. They both want to know WHO DID WHAT and WHERE. Both of ther they are 100 percent ethical) want to keep separate FACTS, FICTION, and LEGEND. Determining which is which is difficult. Any little scrap of information may be the clue that determines which way the ball bounces. So the geneologist never, never throws anything away, including notes on the source of each bit of information. Just like in computer programming, documenting facts is important; especially if you are trying to qualify for the D.A.R. or Sons of the Confederacy.

So what happens to the researcher? In nothing flat even the casual geneologist has collected a stack of notes that would choke a whale. Trying a manual sort, expecially when notes are maybe written on whatever happened to be handy at the time they were taken, soon becomes an impossible chore. This is where Pro/File comes to the rescue.

Collect all your notes and just start typing. Once you have a batch entered into the program, let Pro/File run the searches by name, rank, serial number, place, date, whatever.

I have a lot of information on taped interviews with elderly relatives. This can be coded into Pro/File and used to index a name to a particular cassette. You might want to use a simple "generation" code (gen1, gen2, gen3, etc.) to organize a printout of all ancestors and their descendants. Or how about a church code so you can segregate all Methodists from Baptists! Another code might be used to identify 2nd, 3rd, etc. marriages, or children thereof. Ask your librarian to show you how to use the new computerized census indexes, and the "SOUNDEX" file. The flexibility of Pro/File will allow you to use just about anything for a search command, so use your imagination!

HOW THE PROGRAM WORKS
A guide to the listing

It is virtually impossible for a programmer like me to provide a computer user like you with a program that measures up to your specifications completely. Everyone's needs are different. The nice thing about working with computers is that you can adapt, modify, or jazz up a program very easily if you know how.

This guide, therefore, serves to explain the inner workings of Pro/File 2068. Use it to learn how individual program lines affect Pro/File's operation, and how each command "relates" to every other. Once you have a good working knowledge of how the program works, you'll be able to open Pro/File 2068 up for surgery. You'll be able to cut and hack, modify and improve the program until you have it doing exactly what you want.

This explanation is a traveler's guide to the program listing--not a stand alone treatise on file managers. The text will serve you best if you use it in conjunction with the program listing itself. Watch the Pro/File in action as you proceed.

You'll notice that the text follows the natural flow of the program's operation. Occasionally, a program line will be skipped by the text. This will happen when the line in question does not directly affect the program's operation in the mode or situation currently being described. The function of the line will be covered later when it really performs a significant task.

BASIC vs MACHINE CODE
What does what

The program listing in Appendix I is the Pro/File BASIC. This is what you see when you break out of the program and type LIST. Pro/File's machine code (found in Appendix II) is slightly less visible but it is there in your computer's memory never the less. Pro/File's machine code is located above the normal BASIC memory, 2046 bytes starting with address 63488 and going upwards, are reserved for machine language instructions.

Pro/File 2068 blends BASIC and machine code so that the time consuming search and edit routines are carried out in fast machine language while menus, prompts, and inputs are carried out in BASIC. Generally, Pro/File uses BASIC to set flags and create conditions that enable the machine code to function as it should. The result of this bilingual approach is fast processing times without being totally locked into a machine code program. The BASIC gives you the programmer a flexible modifiable program, and you still have the benefit of blinding machine code speed.

THE CASSETTE LOADING PROCEDURE

When you power up the computer and load the Pro/File cassette, a short program loads which prints the title and copyright notice. This program also performs three functions vital to the operation of Pro/File 2068.

First, RAMTOP is changed using the command CLEAR 63487. This fools the computer into thinking there is less memory available than there really is. All memory from address 63488 to 65535 is thus not used by the computer's BASIC language and is, instead, free to be used for other purposes--in this case to hold Pro/File's machine code and system variables.

The second thing the short loader program does is load the actual machine code into the space just cleared. The instruction:

LOAD "p/f"CODE 63488,2046

loads 2046 consecutive bytes from the tape into memory. The first byte to be loaded is address 63488.

Finally Pro/File's BASIC instructions are loaded. The recording was made using the SAVE "pro/file" LINE 9996 command. Whenever you:

SAVE "program" LINE xx

the program will, when loaded, automatically start executing at line xx. In this instance, line 9996 is the first program line executed.

HOW TO BREAK INTO THE LOADER PROGRAM

If you have a penchant for breaking into program listings in order to glean ideas for your own programs, you can break into this one by holding your fingers on SHIFT and BREAK until the report "D Break" appears on the TV screen. Do this as soon as the "LOADING" sign starts to flash.

Now if you press LIST and ENTER you should get the listing. If you try this and nothing happens, even after several repeated LIST's and ENTER's, you have been fooled by the oldest program hiding trick in the book--that of making the INK color the same as the PAPER color. Whether you write on the TS2068 or on paper, the rule is the same: black on black doesn't show.

To make the program visible type INK 7. Then press ENTER twice. This is what you'll see:

```
   1 BORDER 0
   2 PAPER 0
  10 CLEAR 63487
  30 PRINT AT 5,8; PAPER 1; INK
7;"* PRO/FILE 2068 *";
  40 PRINT AT 7,4; INK 7;"© 1984
 By THOMAS B. WOODS";AT 10,11; I
NK 6;"P.O. Box 64";AT 11,7; INK
6;"Jefferson, NH 03583";AT 19,7;
 PAPER 1; INK 6; FLASH 1;"LOADIN
G"; FLASH 0; INK 6; PAPER 0;" Pl
ease wait";AT 0,0: LOAD "p/f"COD
E 63488,2046
  50 LOAD "Pro/file"
```

PRO/FILE 2068 INITIALIZATION

The loader program replaces itself with the Pro/File 2068 BASIC and begins running at line 9996. This line creates the memory space necessary to store information by dimensioning a large array of 28,020 blank characters. This is done by the first statement:

```
DIM d$(28020)
```

When you add files to the program, d$ is where they are kept. Much more will be written about this array, but for now, it is sufficient to note that the array at this point consists of 28,020 blank spaces and the array is the first variable held in your computer's memory.

The next variable to be set up is a number referred to as "P". This variable serves as a "pointer" that indicates how many of the d$ characters actually hold file information. It "points" to the first unused character of d$. P starts off with the value of 20 because the next portion of line 9996 creates the first file held in d$: 20 characters which say-

```
*SEARCH IS COMPLETE*
```

Just why the first file says what it does will be explained in the discussion on how files are found and displayed.

Two more string variables, a$ and c$, which control the printer parameters are set to their initial values. Each character of c$ represents a line number of a file to be printed. The FOR... NEXT... loop programmed into line 9996 creates a c$ consisting of 15 characters. The first has a code of 1; the second, a code of 2; the third, of 3; and soforth up to character number 15.

The final variable in 9996 is "S". This points to individual file lines when the computer displays files alphabetically.

Lines 9997 and 9998 print the "LOAD" or "CREATE" option which appears after you load the Pro/File tape. Your option goes into Y$. If you type a capital "L", the program jumps to line 5510 to LOAD an existing file. If you type "C" or any key except "L", your are then asked to input a name to attach to this newly created data base. This name goes into F$ at line 9998. F$ remains the same throughout the program. It serves as a "label" to identify a particular file. When you SAVE your updated file on cassette, it will be saved under the name held in F$. Similarly, all LOAD procedures will request a file name to take from a tape--the name held in F$.

When the TS2068 saves or loads, the program name must not be more than 10 characters long. Although it is possible to load a nameless program, the computer will not permit saving a program without a name. Hence the extra checks in line 9998:

IF f$=" " OR LEN f$>10 THEN GOTO 9998

If you press just ENTER or if what you type is longer than 10 characters, go back to line 9998 to re-input a new name. Once an entry passes this test, line 9999 is executed--GOTO line 1--but there is no line 1. In Timex Sinclair BASIC it is quite legal to tell the computer to GOTO a non-existent line number. What happens is the computer executes the first line it finds after the number it was directed to which in this case is line 5--the first line of the Pro/File 2068 BASIC--and the beginning of the display of Pro/File's Main Menu.

FUNCTIONS OF THE MAIN MENU

Here at line 5, border and paper colors are set to black (or zero). The instruction RANDOMIZE p is executed, followed by two POKE commands. This threesome works together to place the data pointer (p) into a two byte integer located at addresses 23670 and 23671, the system variable SEED. Then, the numbers found there are POKED into two more memory locations at 64035 and 64036. This pair of addresses serves as a Pro/File system variable which is used by various machine code routines. They hold the value of p, the data pointer.

The Timex owners manual explains that the RANDOMIZE command sets the system variable SEED so as to hold the number accompanying the command if it is non-zero. Thus when you RANDOMIZE 20 which is what the p variable now equals, SEED, located at addresses 23670 and 23671 store the number 20. You can see how this works by typing into your computer:

RANDOMIZE (some number--not zero)

Then check the value held in SEED. Type PRINT PEEK 23670+256*PEEK 23671 The result will be equal to the number you RANDOMIZED. The RANDOMIZE command used like this is a convenient way to fit large numbers into 2 byte machine code variables.

Anyway, after p's value is stored, two more variables, A and Y are both set to zero. A and Y control Pro/File's autosearch and printer output modes. When A=0 the autosearch function is deactivated. When Y=0 the printer mode is turned off.

Line 5 also dimensions an E$ array consisting of 15 separate strings each with 32 characters. When you look up a file, it goes into E$ at the same time it is printed on the TV screen. The E$ array is a buffer which stores a found file or one that is on the screen. Files can be a total of 15 lines long with up to 32 characters per line.

Next a machine code routine is executed: USR 64268. This routine scans the data stored in the program looking for zeros.

If one is found it changes the zero to a 42. This is done to correct certain file markers that are changed by the alphabetizing function. The reasons why and the explanation of how it is accomplished will be covered in detail later.

Now Pro/File's Main Menu gets printed by lines 20-50. Two subroutines help out in this. The first one, GOSUB 9830, draws a rectangle around the top portion of the menu. The second subroutine at 9850 prints the "control status" and various conditions the program is in. This "status" routine is called at several points in the program to display:

*SPACE OPEN (the length of D$ less P--the number of bytes used)
*FILE NAME (what you typed into F$)
*ORDER (line number by which files are alphabetized)
*PRINT FORMAT (which file lines are fed to the printer)

Note that instructions in lines 30 and 40 set up the INK color and the print position (by PLOT in line 30 and PRINT AT in 40) before the subroutine is called. In this way, the same routine can print at any screen coordinate from any portion of the program.

The advantage of this becomes apparent in the next group of program lines, 100 to 115, which let you input a search command or initiate a special function.

After you input your command at line 100, the next program line comes into play if you type DEFP. Here, the print position is first set to line 14, column 0. The plot position is given the coordinates 120 pixels over and 70 up. Then the computer GOSUBS to 6500 where you are instructed to input your desired print format. Once again the subroutine at 9830 is called by 6500 to draw a rectangle, but this time the rectangle is drawn in a different portion of the screen.

Lines 6510 to 6540 go through rather complex gyrations that allow you to input your printer parameters and check what you input to be sure that you did it correctly. Read the section on printer output to learn what goes on here. Two string variables, A$ and C$, store the print format. A$ stores it such that you can understand which lines get lprinted; C$ stores it so the computer understands. Therefore, when the subroutine at 6500 returns to line 102, A$ displays the print format in the Main Menu.

E$(1) follows the printing of A$ being repeated three times. Recall that line 5 dimensioned the E$ array. This created 15 strings each containing 32 blank spaces. When line 102 prints E$(1) three times in a row, it simply overwrites what was on the screen with 3 lines of spaces. These three

blank lines, therefore, erase the DEFP instructions that were there before.

If you type anything except "DEFP" into X$, the computer skips past 102 to line 103 where if you typed "AUTO" the computer branches to 5200. Since the Auto Search mode does so many different things, it is given its own chapter further on. Suffice it to say that if you do not type "AUTO" the computer goes right past line 103 to 105 where Ink is changed to 5 (cyan), the TV screen is cleared and X$ is checked again to see if you typed "A" for ADD. If you did, you go to line 5000. Like the Auto Search, if you don't type "A", the computer proceeds to 106 and 107 where checks are made to see if you typed "LOAD" or "SAVE". If not, 2 final check points stand between X$ and the search routine.

First, line 110 sends the computer back to line 10 if X$ is empty. If you press just enter when you input X$ this is the case. The search routine must have something to search for so if you don't type anything, its back to line 10 to try again.

The final check makes sure you don't type too many characters and that your syntax is correct. Although the search routine can accomodate any number of different words separated by the token "AND", a search command that is longer than 32 characters or one that ends with the "AND" delineator will send the computer into a tail spin. Line 115 looks for these occurrences ar sends you back to line 10 to re-input a new search command if it finds one.

FINDING AND DISPLAYING FILES

The next group of 5 lines (119 to 1010) is the powerhouse of the whole program. These lines take the word that you input as a search command and match it against the data stored in the files. When a match is found, the file containing the match is pulled from the data array (D$) and simultaneously placed on the TV screen and into the E$ array. Machine language programming is called upon to do the bulk of the Search and Print functions. This dramatically improves the speed of operation over a similar routine written entirely in BASIC.

The order of improvement depends in large part on what data you have and what you search for, but typically machine code could find and print a file in a second or less while BASIC would take 20 seconds to several minutes to perform the same job.

To understand how the search routine works a brief review of what the computer has done thus far is in order. From the Main Menu you input a word at line 100. The checks that follow in lines 102 to 115 either send the com-

puter back to the Main Menu or off to perform a special function if you typed a special word or if it was improperly entered. If you do not imput the word DEFP, A, AUTO, LOAD, or SAVE; and what you type is less than 32 characters; and the last character is not the token "AND"; the computer treats the word you enter as a search command and line 119 is executed:

```
LET X$=X$
```

On the face of it, this line does nothing. It takes the word you input and makes it equal to itself. Not very impressive. The line does more, however. The computer uses an area of memory to store various numbers which serve as pointers and condition flags. This is called the SYSTEM VARIABLES area of memory. (see page 262 of your Timex manual for a description of these variables) One such variable, called DEST, stores the address in memory where a string which is being operated on is located. LET X$=X$ causes the computer to store the address of X$, the search command, in DEST (23629 and 23630). It also stores the length of X$ in another system variable called STRLEN (23666 and 23667). Line 119's purpose, then, is to set up these two system variables so they contain useful information for the machine language which is executed by the next program line.

Line 120, RANDOMIZE USR 64040 causes the computer to depart from BASIC and execute machine language instructions beginning at address 64040. This routine makes a copy of X$ and places it in a 33 character search command buffer--which I term SBUF--that starts at address 63970.

If you typed the word "FICTION" as a search command, USR 64040 would copy this word into the buffer. Address 63970 would hold an "F"; 63971 would hold an "I"; 63972, a "C"; and so forth.

USR 64040 also sets 3 of Pro/file's own system variables--DCNT, SADR, and DPTR to their initial values for a search:

DCNT (pronounced "D-count") counts the number of bytes of data to search. At the beginning of every search, DCNT equals the BASIC variable P, the total number of bytes used in files.

SADR (pronounced "Search Address") points to the address of the current command word in the search buffer. SADR starts off being equal to 63970--the start of SBUF.

DPTR (pronounced "D-pointer) points to the address in memory of the last found file. Since no file has yet been found (we haven't even started to search yet), DPTR points to the first file in memory. Recall that it says "*SEARCH IS COMPLETE*"

After the search command is placed in SBUF, a special character is placed at the end of the word. This character is the "\" or CHR$ 92. It tells the computer that it has reached the end of the search command when a search is being made.

This machine code has one last feature that performs no significant function to the program as it is written, but it provides you the programmer with a means to modify Pro/File 2068 so it can perform in programmable search modes. The secret to this "enhanced" mode lies in the use of RANDOMIZE to call the machine code. Remember that RANDOMIZE sets the system variable SEED to a number used in conjunction with the command. USR 64040 looks to see if you placed a "\" in the second to the last spot of the search command. If you did, the last character of the search command will be stored in address 23670--the second byte of the SEED variable. If you did not, 23670 will always be equal to 1. Therefore, a search command such as:

FEBRUARY 17

will result in PEEK 23670 equal to 1. But a search for:

FEBRUARY 17\A

will produce a PEEK 23670 equal to 65 which is the code number for the letter "A".

In both cases the search will be just for FEBRUARY 17, but in the latter a flag is raised. Address 23670 holds a special number. You could add a program line which PEEKs 23670 and if it meets a certain requirement, your program line could branch out of Pro/File and into your own block of programming. Examples of how this might be accomplished will be covered in the chapter on how to modify Pro/File 2068.

Line 150, LET B=USR 64101 performs the actual search. Here's how: First the computer takes the first character of D$ as given in DPTR and matches it against the first character held in the search buffer as given in SADR. If a match occurs, the second character of each is compared. The progression of matching characters continues until one of 3 conditions occurs. If the matching sequence finds a non-match it means that the word held in d$ is not the same as the word in SBUF. The program starts looking for the first character of the search buffer again.

If the computer finds the token, AND, in the buffer, it means that one word of a multi-word search was found in the data. The program marks the start and end points of the file the match was found in, then it scans the file for the next word. As long as matches are found and "AND" tokens are encountered, the program repeats scanning the file for matches to the next word.

The program counts each byta of data as it is compared. It starts at the numbe of bytes used for data and decrements. When it reaches zero, the computer knows that all the files have been checked. This is the third condition that stops the matching process. In other words, when the counter is zero, all files have been checked. Break from the search and indicate to the operator that the search is complete.

Pro/File 2068's search routine is written so that when this counter reaches zero, the first file in memory is displayed. This is why when the variables were first initialized in line 9996 the first file, D$(1 TO 20), was programmed to hold the words "*SEARCH IS COMPLETE*".

As long as you have data stored in the program and the search buffer holds a word that also appears in a file, the search progression will ultimately come to the "\" or the last character of the search command. This signifies that the word or words in the search buffer is also held in a file. The computer then stores the address of the first found file character in another Pro/File system variable called FADR (address 64026 and 64027). DPTR and DCNT store the location in the data of the last character matched. Then the machine code returns to BASIC.

Note that line 150 creates a BASIC variable B. As long as a search has not progressed to the end of data, B will equal the number of characters left unchecked. When the search is complete, B equals P, the total number of bytes used for data.

In either case, when a search finds a file in which the search command is present, or when all the data has been scanned and no match was found, a files starting address will be stored in FADR before the machine code returns to BASIC. FADR will point to either the file containing the match or the first file in memory which says "*SEARCH IS COMPLETE*".

Now the BASIC line 1000 is executed. First, the command PRINT AT 0,0; sets the print position at the top of the screen. Second, the E$ array is redimmed. This has the effect of erasing anything that might have been present in it. After execution of this command, E$ once again becomes an array of 15 strings each containing 32 blank spaces. Third, LET E$(1)=" " acts much like LET X$=X$ mentioned earlier. It causes the 2068 system variable DEST to point to the address of the first character of E$(1). Finally more machine code (USR 63575) transfers the found file into the E$ array and prints it on the TV screen at the same time. After USR 63575

is executed, E$ contains the file whose address is pointed to by FADR.

The next line, 1010:

```
IF e$(1 TO 18)=d$(2 TO 19) AND b<>P THEN CLS:
```

compares the file printed on the screen against the SEARCH IS COMPLETE file which is located in the second to the eighteenth characters of D$. If they match and P is not equal to B (the search is not complete), the computer clears the TV screen and goes back to line 150 to search further. This is a check to make sure that the computer doesn't properly find and print the first file unless the search really is complete. It is possible that a search the computer undertakes could find a match in the first "SEARCH IS COMPLETE" file even though the search has just begun. A search command like "COM" could bring this file on the screen. The computer can tell if this situation occurs because the only time B will equal P is after all data has been scanned. This line protects you from a potentially confusing situation should you enter "SEARCH", "SEA", or some other combination of characters that would match with a portion of the first file.

In English, line 1010 says, "If you see SEARCH IS COMPLETE and all the files have not been scanned, clear the file off the screen and keep looking."

Line 1040 prints the search command at the bottom of the screen just in case you forgot what it was you were looking for.

Line 1045 tests the BASIC variable A. This variable, remember, tells the computer if it is doing an Autosearch. A equals one in the Auto mode--zero otherwise. Therefore, this line directs the computer to 7000 if A is not zero. Autosearch functions will be covered in detail later on, but for now, assume that the Autosearch mode is turned off by A being equal to zero. The result is that 1045 is skipped over by the computer and the program is now ready to print its Display Options.

THE DISPLAY OPTION MENU

Lines 1050 and 1051 print the options given after the display of every file. They also let you input your selection. If you look at 1050, you will see that the first batch of possible commands are listed. About midway down the line is:

```
: INPUT "OPTION? ";y$: IF y$="M" THEN PRINT . . . .
```

This command, thus stops the printing of the first third of the Display Options to let you input your selection. If you type "M" into Y$ the second group of options is printed by the same line and once again 1050 stops to input Y$ a second time.

If you don't type "M" when the computer stops the first time, the machine skips the second display as well as the third which is printed by the execution of line 1051. Instead you go to line 1055 where the computer decides what to do in response to your input.

Line 1051 is carried out only if you type "M" a second time. This line resets paper and ink colors and gosubs to 9850 which you recall is the subroutine that prints the STATUS of such things like OPEN, ORDER, DEFP, etc. After the subroutine returns, line 1051 stops to input Y$ yet a third time. Now if you type "M", you go back to 1050 to repeat the whole display again. If you make a different selection, the computer continues to 1055.

These two Display Option lines (1050 and 1051) therefore, will recycle as long as you keep typing "M". As soon as you type something else, its off to process what you typed. The "Option" lines are capable of displaying 3 different sets of displays. Regardless of what may already be on the TV, its only the "M" selection which displays another group of commands. Type any other selection independent of what the display actually prints on the screen, and the computer continues out of the Options to process your selection.

Lines 1055 to 2000 branch the computer to different functions depending on what you type into Y$. Line 1060:

```
IF y$=" " AND b<P THEN CLS : GOTO 150
```

is executed if you type just ENTER and if B<P or put another way:
If the search is not complete, clear the TV screen and go to line 150. This causes the computer to continue searching the D$ array for more matches to the search command. Following the Display Option Menu, press ENTER to continue. Line 1060 is the line that actually makes it happen.

Line 1070 looks for the letter "N" meaning "start NEW from the Main Menu". If "N" is found in Y$ then go back to line 1. This displays the Main Menu and lets you input a new search command.

If you don't type "N" or just ENTER, the program comes to line 1075 which executes USR 64268. This is the same machine code routine that was called by line 5. During an alphabetic sort certain pointers in the files are changed and must be restored to their original values. USR 64268 is the routine that does the trick.

Read about USR 64268's why's and wherefore's in the section on alpha-numeric sorting. Its interesting to note that the machine code called by this line is executed by an IF... THEN.... statement:

```
IF USR 64268 THEN
```

may look a bit strange, but it is absolutely legal and syntactically correct. The only thing the line does is evaluate the expression USR 64268 by running the machine code. Then the computer goes on to the next line without acting on the evaluation like the more common IF.... THEN.... statement would.

Normally this kind of program line would appear like:

```
IF X=1 THEN GO TO 1000
```

where the expression X=1 is evaluated and if it is true, GO TO 1000. If it is false, continue to the next line. In the case of Pro/File line 1075, USR 64268 is evaluated but the next program line is executed regardless of whether the computer finds the expression true or false.

This method of running machine code is an excellent way to do it if you do not want to affect any other variables in the program. You could just as easily execute the code by using RANDOMIZE USR 64268, but this changes the value held in the SEED system variable. Remember that the programmable search function leaves a number in SEED which you might want to use if you modify the program. Special attention was given in Pro/File's design to insure that SEED would not be changed during a search.

The EDIT or DELETE modes are actuated by line 1080. Here, two conditions must be met before altering files is possible. First:

```
PEEK 64026+256*PEEK 64027 must not equal
PEEK 23627+256*PEEK 23628+6
```

When the search routine was discussed Pro/File's FADR (64026 and 64027) variable was mentioned. This variable holds the address of the file to be displayed. The computer has a system variable of its own called VARS (23627 and 23628) which holds the address of the start of the BASIC variables area of memory. The D$ array is the first variable held and its first character always begins 6 bytes above the address held in VARS.

Since the first file in D$ (the SEARCH IS COMPLETE file) is thus found 6 bytes above the address held in VARS, line 1080 says that in order to EDIT or DELETE, FADR must not be pointing to the first file and you must be pressing the "E" or "D" key.

If both conditions are met the computer goes to line 4000 to start the edit process. Line 1080, therefore, edits a file if you press "D" or "E" of the Display Options but only if SEARCH IS COMPLETE is not printed on the TV.

Repeating a particular search is carried out by line 1090 if you type "R" of the Display Options. First the screen is cleared to erase the file currently displayed. Then LET X$=X$ causes the search command held in X$ to be pointed to by the 2068 system variable DEST. Finally, the computer goes to one of two line numbers depending on whether or not the Autosearch function is turned on (A=1 if yes, A=0 if not). The line says:

```
GOTO 119+(A*5121)
```

Therefore, if A is one (Autosearch on) the computer jumps to line:

```
119+(1*5121)   or line 5240
```

If A is zero (Autosearch off) it goes to:

```
119+(0*5121)   or line 119
```

If the computer just heads straight to line 119 the entire search sequence is carried out as if you typed the search command from the Main Menu. The contents of X$ (what you last entered as a search command) are not changed so in effect, you make the same search starting from the beginning of D$.

If A is one and you go to 5240, the computer does the same thing except line 5240 first resets 2 alphabetizing reference strings (P$ and Q$) to their initial values. Then it goes to 119.

Display Option "C" (copy the file on the printer) will cause line 1100 to be executed: GO TO 7205 to lprint the file held in the E$ display buffer.

The commands accessible from the Main Menu (ADD, SAVE, LOAD, DEFP, or creating a new search command) are made available to to the Display Option menu by lines 1200 and 1500.

At 1200, if you type "DEFP" the command:

```
PRINT AT 20,0,,,,
```

is executed first. This causes the bottom 2 lines on the TV to be erased. Next, print and plot positions are preset for the DEFP subroutine which follows. Execution is the same as when it's run from the Main Menu. After this routine returns, INK is changed to zero (black) and the rectangle drawing subroutine is called (GOSUB 9830). Because PAPER and INK are both the same color, drawing the rectangle this time causes the one that is already on the screen to be erased. After that Y$ is changed from "DEFP" which you just typed to "M" so that after the next command is carried out (GOTO 1051) the status of the newly altered print format will be printed on the screen. All this is done without erasing or altering the E$ array or the file which is displayed. It is therefore possible to change print format and lprint the same file as many times as you need.

Line 1500 brings the other Main Menu commands to the Display Option repertoire. If what you type into Y$ is not just ENTER, "E", or "D", this line makes X$ (the search command) the same as Y$. Then the computer jumps back to 105 which is the beginning of the X$ analysis. Here, the machine checks for the ADD, LOAD, or SAVE options. Everything is the same as if you typed the command from the Main Menu. If Y$ and thus X$ too are not equal to "A", "LOAD", or "SAVE", the contents of X$ are treated as a search command and the program recycles through the entire search, print, and Display Option sequence.

The lines that process the display option selection (1050 to 1500) cover every imaginable situation except for three. If the SEARCH IS COMPLETE file is on the TV screen selections of "E" for EDIT, "D" for DELETE, or just ENTER to continue searching will not be processed and line 2000, GOTO 1050, is executed. This brings the computer right back to start the Display Option procedure again. In other words, "E", "D", and ENTER are illegal moves when SEARCH IS COMPLETE is displayed. Line 2000 refuses to allow these entries and makes you go back to the beginning of the Display Option menu to redo your selection until you get it right.

HOW FILES ARE ADDED

If you type "A" either from the Main or Display Menus, line 105 branches to the FILE ADD routine beginning at line 5000. Here, E$ is redimmed so it becomes all spaces. Next four more Basic variables are assigned values:

LET M=0	This is a flag that helps determine which of three EDIT command menus will be displayed. M will always be either 0, 1, or -1.
LET Z=23658	This is an address in the system variables area of memory that corresponds to FLAGS2, a flag which controls the capital or lower case modes when printing letters on the screen.
LET I=0	A variable that governs whether letters typed on the screen will overwrite what is already there or INSERT between 2 characters on a line. INSERT mode is ON when I=1. Its OFF when I=0.
LET L=0	The line number on which the cursor blinks.
LET C=0	The column number of the cursor. Initial values of L and C are zero, hence the cursor starts blinking at line 0, column 0--the top left corner of the screen.

Because the E$ array is configured as 15 lines of 32 columns each, it overlays an actual display of a file quite nicely. The variables L and C are used to describe individual characters in the array as well as what's on the screen. There is one subtle difference however.

When you deal with a position on the TV, you can refer to line zero or column zero. Not so with strings or arrays. If you set up a string to equal the characters:

HELLO THERE COMPUTER FANATICS

you could not refer to character number zero. If you try you get an error 3--subscript wrong. The solution to overcome this discrepancy is simple. The variables L and C point to the true line and column numbers on the screen, but when the computer puts the character AT L,C into the array it puts it into E$(L+1, C+1). Therefore, the character in the array found at E$(1,1) can be found on the TV screen at 0,0. The array character at E$(12,5) would be printed at 11,4. The trick may sound confusing at first, but I assure you this stems from my crude way of putting it. For the computer it's a snap.

At line 5004 the first of three displays is printed which list various commands which can be implemented. Then 5005 starts the edit cursor blinking. The command:

PRINT AT L,C; SCREEN$ (L,C)

tells the computer to print at line L, column C; the screen character it finds at line L, column C--not very interesting except that when FLASH is set to 1, the computer flashes the character. A blinking cursor is born.

The next line, 5007 implements the ON ERR GO TO command. In this case, GO TO 5005. This command turns off the normal error stopping procedure. Whenever an error occurs, the computer is instructed to go to line 5005 rather than stop as it should with a report code. Actally this whole ADD/EDIT part of the program is written so that no errors could occur except if you press the BREAK keys. This is the sole reason why ON ERR is used--to prevent an unintentional BREAK while adding or editing a file. If BREAK is pressed, you go to re-blink the cursor at 5005 instead of halt the program.

Line 5010 watches the keyboard and puts the computer on hold until a key is pressed. It says:

IF INKEY$=" " THEN GOTO 5010

or put differently, watch for a keypress. If no one is pressing a key then watch some more. The line is like the trigger of a gun. The computer sits poised until you touch a key. Then off it goes to execute the lines that follow.

Line 5016 looks for a special condition--that of pressing STOP (symbol shift and "A" simultaneously). This is the line that breaks out of the ADD/EDIT mode. If you press the STOP key, normal error handling is restored to the program, the flashing cursor is turned off, and the computer is directed to line 6000.

As long as STOP is not pressed, the computer cycles right past this line to 5020 where the key that is being pressed is placed into Y$. Then the same program line prints at screen line L, column C; the SCREEN$ (L,C) --the character that happens to be at L,C--in the non-flashing mode. You may wonder why there is a need for a command that appears to print what is already on the screen. The reason is that L and C mark the coordinates of the cursor which flashes the character present. This command prints the character in a non-flashing state because L and C are about to change. The old cursor position must stop blinking. Just where the cursor is supposed to move is determined by the last statement in 5020:

```
IF CODE Y$<16 THEN GOTO 5100+CODE Y$
```

checks the code number of the key you press. If it is less than 16 you go to program line 5100 plus the code representing the key pressed. All codes less than 16 are function codes.

In the back of the Timex manual is a list of characters used by the computer. A look at the first 15 codes listed reveals that they are not used for characters or tokens. Pro/File makes use of these characters by giving them special meaning. That is, if you press a key with a code of less than 16, the computer is told to branch to a different program line to handle the specified requirement. Line 5200 does this by sending the machine to line 5100 plus the code for the key pressed.

For example, the character listed for code 13 is ENTER. Therefore, when the program finds you pressing the ENTER key, it goes to line 5100 + 13 or 5113 where in this instance, the cursor is made to drop down a line and blink in column zero (the equivalent of Carriage Return).

When you look at the chart in the Timex manual, you'll see quite a few codes that do not actually print a character or token, and finding just which key produces a specific code is not evident. You can find out what is where, however by keying in this simple program.

```
10 LET Y$=INKEY$
20 PRINT AT 10,10;CODE Y$;"_ _ _"        (underlines denote SPACES)
30 GO TO 10
```

Run the program. Press different keys and watch their codes get printed on the screen.

Before getting too carried away with character codes and the keys that represent them, let's get back to Pro/File's line 5020. For now, assume that 5020 doesn't find you pressing a "command key". Instead, suppose you press the capital letter "B" (code 66).

Line 5020 places the letter B into Y$. If the INSERT mode is turned off (I=0), line 5025 is not executed. 5025 will run only when I has a value other than zero.

5030 prints the character B on the screen at L,C. As the letter gets printed the line also places "B" in the E$ array (at position L+1,C+1). Then both L and C are advanced to the next column and line if necessary. The statement:

```
LET C=(C+1)*(C<31)
```

increments the column if it is not already at the far right (C=31). The expression C<31 is evaluated by the computer. If the expression is true it is given the value of 1. If not true, the value is 0. Therefore, the statement reads:

```
LET C=(C+1)*1        if C<31
LET C=(C+1)*0        if C=31
```

Since a number multiplied by zero equals zero, C becomes zero whenever the computer tries to advance it past 31. In other words, after a character is printed, advance the column number for the cursor to the next one over, but when you get to the right most column bounce the cursor back to the left hand side.

The line variable L is similarly incremented by the statement:

```
LET L=L+(C=0)
```

Translated, this means whenever C becomes zero, a result of the previous statement, advance L to represent the next line. As before, the expression (C=0) is evaluated as either 1 or 0 depending on whether or not the expression is true or false. The variable L, then, becomes either L+1 or L+0.

The statement that comes next makes sure that L does not become greater than 15, the bottom line available for displaying data. The statement reads:

```
LET L=L*(L<15)
```

As before, (L<15) is given a true/false value. As long as L really is less than 15, L maintains its value because L*1=L. But when L equals 15, L becomes L*0. The cursor moves up to the top line.

Lastly, line 5030 turns the flash back on and prints at the new L and C the screen character found there: flashing.

By now it should be clear that as file characters are printed on the TV, they are also placed into the E$ array. What appears on the top line of the screen is also present in the 32 characters of E$(1). The second line of the TV is in E$(2) and so forth.

What has not been mentioned is the INSERT mode and how it differs from the OVERWRITE mode. If I=0 the preceding actions perform as indicated. They simply ignore what characters used to be on the screen and overwrite them with the new character in Y$. When I=1, things are a bit different. Instead of just blasting away the old letters, the INSERT mode makes a blank space between the location of the cursor and the character that appears directly to its right. Then the computer inserts the letter found in Y$ into this space.

Line 5025 is the line that does the inserting if I=1. At any given moment L will point to the current line while C points to the column pointed to by the cursor. Since it is known that the entire line L can be found in the E$ array, simple string slicing techniques can be used to insert new characters. In 5025 the command:

```
LET E$(L+1)=E$(L+1, TO C)+" "+E$(L+1,C+1 TO )
```

does just this. In plain English, it translates to:

> Let the line of the array on which the cursor blinks equal itself--up to the column number of the cursor; PLUS a blank space; PLUS the remainder of the line of the array from the point after the cursor to the end.

The blank space is where the newly typed character will be printed. Everything in the array on the line in question is shifted to the right one column.

With the array suitably altered, the only thing left to do is change what appears on the TV screen. This is done by the last statement in 5025:

```
PRINT AT L,0;E$(L+1)
```

print the new line from the array (L+1) on line L of the TV.

The variable I is used in line 5040 to select the color of the cursor. After FLASH is turned on, INK is set to 6 plus whatever the current value for I happens to be. Since this variable is either a one or zero, INK will be either six or seven: yellow in the OVER MODE (I=0); or white in the INSERT MODE (I=1).

As in lines 5005, 5020, and 5030, this line prints at L,C; the current character found on the screen at this position. When FLASH is set to 1, the character blinks. Finally, 5040 creates a short delay by running through a "do nothing" FOR.... NEXT.... loop in order to give you time to get your finger off the key.

The next line, 5013, sends the computer back to 5010 to repeat the whole process of:

* Get the character from the keyboard
* Check for a "command" and process if it is
* Put the character in the E$ array and on the TV if it isn't
* Adjust the cursor and flash it
* Repeat the sequence

This cycle runs over again once for every character typed until you press STOP at which time line 5016 breaks you out.

The special "command codes" which cause the computer to execute the various edit functions branch from this cycle at line 5020 where the code number for Y$--the key you press--is checked, and if it is less than 16 you go to line 5100 plus the code for Y$. A review of all the possible edit commands is in order.

THE KEY PRESSED	FUNCTION	CODE
shift-3	LINE DELETE	4
shift-4	LINE ERASE	5
shift-2	CAPS LOCK	6
shift-1	INSERT/OVER	7
left arrow	move cursor left	8
right arrow	move cursor right	9
down arrow	move cursor down	10
up arrow	move cursor up	11
shift-0	DELETE character	12
ENTER	cursor down/left	13
shift-symbol shift	LINE INSERT	14
shift-9	more commands	15

The codes for each of these keys specify the one line which affects the respective function. Since codes range from 4 to 15, jumps to lines 5104 to 5115 take place. What follows is a description of what occurs for each function.

Code 4-LINE DELETE

Line 5104 deletes an entire line by using a FOR.... NEXT.... loop to move each line of E$ from the line specified by the variable L to the last line--E$(15)--up one position. E$(L) becomes E$(L+1), E$(L+1) becomes E$(L+2)

and so forth. At the conclusion of the loop, the 14th and 15th lines in E$ are the same so the instruction LET E$(15)=" " causes the last line to become blank.

To better visualize what line 5104 does, imagine a list of 5 names:

PETER
MARY
SAM
BILL
SUE

In the case of Pro/File 2068 you could, of course, have a maximum of 15 names or lines of any other information. To delete Sam's name from the list you would first erase it and then write BILL where Sam used to be on line 3. Next, you would erase the BILL that remains on line 4 and write SUE. Finally, since SUE is written twice on lines 4 and 5, you erase line 5. This leaves you with 4 names. SAM was deleted. Every name after the one that was deleted is moved up one line and the last line is erased.

Code 5-LINE ERASE

Line 5105 acts similarly to the delete function in that the line specified by the flashing cursor is blotted out. However, the ERASE function leaves the lines that follow in their original positions. Also, only the data that appears to the right of the cursor is erased. Anything to the left of the cursor remains on the screen.

The LINE ERASE procedure is much simpler than deleting. 5105 says:

```
LET E$(L+1, C+1 TO )=" ": PRINT AT L,0;E$(L+1): GO TO 5040
```

Translated, this means: let the characters from the cursor to the end of the E$ printed on line L equal all spaces. Then print on line L all of this new E$. Then go to line 5040 to continue the edit process.

Code 6-CAPS LOCK

CAPS LOCK (or UNLOCK) uses the variable, Z, to point to the system flag located at address 23658. This flag specifies whether upper or lower case letters are to be printed. If 23658 holds a zero, lower case characters are printed. If PEEK 23658 equals 8, it is upper case. Line 5106 pokes address 23658 to either a zero or an eight, alternating from one to another with each press of the SHIFT-2. The command:

```
POKE Z, (PEEK Z=0)*8
```

changes the value held in address Z (read that 23658) by evaluating the expression, PEEK Z=0. IF PEEK Z is equal to zero, the expression is true and thus has a value of 1. Therefore, Z is poked to (1*8) or 8. If PEEK Z turns out to be eight, the expression is false. The expression then has a truth value of zero. In this case the computer would poke Z to (0*8) or 0. Another way to write this would be:

Place a zero into address Z if address Z holds an eight.
Place an eight into address Z if address Z holds a zero.

Line 5106 also momentarily prints either a ">" or "<" to indicate which mode the computer is in. This is done by the command:

PRINT CHR$ (60+(PEEK Z/4))

The codes for ">" and "<" are 62 and 60 respectively. If PEEK Z is zero (lower case), the command prints:

CHR$ (60+0) or the "<"

If PEEK Z is eight (upper case) the command prints:

CHR$ (60+ 8/4) or CHR$ 62, the ">"

After the sign is printed, the computer pauses briefly, prints the original file character that was in the cursor position, and goes back to line 5040.

Code 7-INSERT/OVER switch

Handling this function is simple. The only requirement is that the variable, I, be alternated between 1 and 0.

LET I=NOT I in line 5107 does this

Code 8-CURSOR LEFT

To move the cursor left one position, the variable C must be decreased, but the computer must take care not to reduce C's value lower than zero or it wouldn't know where to print the cursor. The range must always be from 0 to 31. Line 5108:

LET C=C-(C>0)

decrements C as long as the variable is greater than zero. If it is zero, C is left unchanged (LET C=C-0). This represents the left most column of the TV screen.

Code 9-CURSOR RIGHT

This function increases the value of C as long as it is not at the far right of the screen (C=31). Line 5109 accomplishes this:

```
LET C=C+(C<31)
```

In other words, make C equal to C+1 if C is less than 31. Make C equal to C+0 if C is equal to 31.

Code 10-CURSOR DOWN

To move the edit cursor from line to line, the variable, L, must be altered. Increasing its value moves it down. Line 5010:

```
LET L=L+(L<14)
```

works to move the cursor down in the same way that line 5109 moves it over. The expression, (L<14), keeps the cursor from advancing beyond line 14--the 15th line of a file.

Code 11-CURSOR UP

This function decrements L in line 5011. As before, the procedure is the same as moving the cursor to the left except it is L rather than C which is decreased in value.

Code 12-DELETE CHARACTER

Line 5112 alters the E$ pointed to by the line cursor by deleting the character the cursor represents. When the SHIFT and "0" keys are pressed simultaneously, this line is acted upon. The edit cursor is located at line L, column C. The portion of the E$ array printed on line L can be referred to as E$(L+1). The actual character to be deleted would be that which occupies E$(L+1,C+1).

```
LET E$(L+1)=E$(L+1, TO C)+E$(L+1,C+2 TO 31)
```

then, redefines this portion of the E$ array so it equals everything to the left of the cursor plus everything to the right of the cursor. The character at L,C where the cursor flashes is left out. Next, the newly defined line is printed on top of the old one and the computer jumps back to 5108 where the cursor is moved left one column.

Code 13-NEXT LINE

When you press ENTER, the cursor moves down one line and left to column zero. This movement is effected by line 5113 where C is reset to zero and L becomes L+1.

Code 14-LINE INSERT

Inserting lines is just the opposite of deleting an entire line. Instead of moving lines up one position, the line INSERT function moves every line after the cursor down one position and creates a blank line where the cursor flashes. Program line 5114 creates a FOR... NEXT... loop which counts backwards from 15 to L+2 (the numbers representing the last line of the E$ array up to the first line of E$ below the cursor). As the loop repeats, every line of E$ between these points is shifted down. Then, E$(L+1) which is now duplicated on the next line is made blank.

Code 15-MORE COMMANDS

Line 5115 prints a second list of edit commands if you press the SHIFT and 9 keys together. Remember back at the beginning of the ADD/EDIT mode, the variable M, was set to zero. The list of commands here at 5115 will be printed only if M=0. After printing, M is changed to -1 so the next time the computer comes here it passes 5115 and executes 5116 where the now familiar subroutine that prints Pro/File's STATUS is called. Following this, M is given the new value of +1. The next time SHIFT-9 is pressed, neither 5115 or 5116 is executed. Instead, 5117 sends the computer back to 5004 where M is reset to zero. Like the Display Options, two different lists of edit commands are displayed alternately, and in both instances the STATUS subroutine is also available.

CLOSING THE FILE

The only special command which is not handled by lines 5104 to 5117 is that of closing the file. This is accomplished by pressing STOP--symbol shift and the letter A. Instead, the file CLOSE routine starts at 5016 where the computer restores its normal error stopping procedure, turns off the flash, and jumps to line 6000 where each line of E$ is transferred into the D$ storage array.

The E$ to D$ transfer is accomplished by means of a machine code routine, USR 63489. Each line of E$ is added separately. Program line 6004:

```
LET E$(X)=E$(X)
```

causes the system variable, DEST (address 23629), to store the address in memory of the current line of E$ being added. The machine code USR 63489 then loads the characters beginning at this address into the next unused bytes of D$.

USR 63489 does this character by character, line by line. Since each line of the E$ array holds 32 characters, the machine code could repeat the transfer of letters on a line 32 times if all 32 columns actually hold something. If only a few characters are present on a line, however, the code chops off all trailing spaces and loads the byte of D$ immediately following the last character of the line with a "1". This serves as a delineator of file lines. When, in a search, a file is pulled from D$ and displayed on the TV screen, it is this CODE 1 that tells the computer to print the characters which follow on a lower line.

If an entire line is blank, the computer puts a second "1" right after the first to indicate a completely blank line. Note that this occurs for all 15 lines of E$ even if a file consists of just 2 or 3 lines of data. Therefore, even if files hold just a few characters on one or two lines, 15 code "1" characters will be used in the D$ array.

After all 15 lines are transferred, the Basic line 6020:

```
LET P=USR 63530
```

does two things. First, the total number of characters used in D$ is tallied and the result is placed in the Pro/File 2068 variable PPTR (address 64035). Second, the last byte used in D$ is given the value of 42. This corresponds to the asterisk.

Like the CODE 1 delineators noted above which separate lines within a file, the CODE 42, or the asterisk (*) separates one complete file from the next. Before USR 63530 returns to Basic, the BC register pair is loaded with the value held in PPTR. Thus, when the program says, LET P=USR 63530, the value of BC is placed in the Basic variable P. In this way the variable P always holds the current number of bytes used to hold your files.

After the data transfer, line 6030 sends the computer back to line 1 where the Main Menu is placed on the screen.

HOW FILES ARE DELETED

If you make a search for a file and decide to press "D" of the Display Options, the computer jumps to 4000 where files are deleted. Here, another machine code routine, USR 63640, erases the file from D$.

To understand what takes place when a file is deleted remember that a file must be searched before anything can be done to it. Once a file is

found, Pro/File's system variables point to key information used by the deletion routine:

FADR points to the address of the first character of the file
FEND points to the last character of the file
DCNT stores the number of bytes of data that appear after the found file
PPTR stores the total number of bytes used for data

USR 63640 calculates the number of bytes used for the found file by subtracting FADR from FEND. Then it moves every character after the address held in FEND up this number of bytes. The operation is a machine code equivalent of the Line Delete function done in Basic when you edit a file. In this machine code deletion routine, it is DCNT which tells the computer how many bytes to move. The difference between FEND and FADR is subtracted from the value stored in PPTR so it will reflect the new number of bytes used. This value is present in the BC register pair on return to Basic. When the Basic line 4000 is executed:

```
LET P=P-USR 63640
```

this value is subtracted from P to update it as well.

Lastly, 4000 sends you back to the Main Menu at line 1 if it was "D" that you typed into Y$ when you made your Display Option selection.

HOW FILES ARE CHANGED

The reason why this condition "D" of the options must be specified before line 4000 jumps to 1 is because the "E" for EDIT option will also come to the file deletion routine at 4000. In the case of editing, however, the computer must do more than just delete. Therefore, if Y$ equals "E", rather than "D", the next line 4010, makes the computer GO TO 5002.

You may wonder why a file gets deleted every time it is altered. A file located somewhere in D$ is most likely sandwiched between other files. The space it occupies can easily be determined by subtracting FADR from FEND--the address of the start of the found file from the address of its end.

When you edit a file, the length will quite possibly be different after the changes are made. In order to open up enough space to fit the enlarged file or to shrink the excess in the case of a condensed file, it would be necessary to program some rather complex "shuffling" routines. Instead Pro/File 2068 avoids this additional complication by automatically deleting the file, accepting any changes you make, then adding the "new" file back on to the end of D$. When this approach is followed, "shuffling" is not necessary because the file you're editing is no longer sandwiched between other data.

It is pulled out of D$ completely, but it is still present in memory. The E$ array has a copy of it. When you change the file, you use the regular add routine that works with E$. Since adding and deleting functions are already built into the program, they can be called upon to edit existing files without the need for additional programming.

So, if you press "E" of the Display Options, the file printed on the screen is deleted from D$. Then the Basic line 4010 sends the computer to line 5002. This is the file add part of the program discussed earlier. Note that when you add a file you go to 5000 where E$ is dimensioned. When you edit a file the computer cuts in just after this line. E$ holds a copy of the found file that was created when it was printed. Thus, even though the file was deleted from D$, you still have it in E$ and on the TV screen. These conditions make it possible to "Edit" a file using the "File ADD" routine. Every function is the same.

AUTOSEARCHES

When you type "AUTO" at the Main Menu, line 105 sends the computer to 5200. Here the variable, A, is set to 1. This variable acts as a flag that tells the computer whether it is making an AUTOSEARCH (A=1) or a regular search (A=0). Next, line 5200 asks if you wish to send files to the printer. The commands:

```
INPUT Y$: LET Y=Y$="Y"
```

sets the numeric variable, Y, to either a 1 or a 0. If you Enter a "Y" when the computer asks if you want a print out, the expression Y$="Y" is true. This means "1" to the computer and Y becomes 1. Type any other key and Y will equal 0. The printer will be activated only when Y=1.

Line 5210 asks you to input a Search Command into X$. Throughout the program the sole function of X$ is to store Search Commands. The action here is no different than when you input a command from the Main Menu. The last statement in 5210 traps any invalid entries and provides a means to escape the AUTOSEARCH MODE by jumping back to the Main Menu. If you press just ENTER or if what you type is longer than 32 characters you go back to line 1.

Assuming you did enter a bona fide command, the next thing the computer does is ask you to type which line you wish to use as the basis for alphabetizing. Whatever you type goes into Z$ at line 5220. As in the previous line, if you type just ENTER you break out of the AUTOSEARCH mode and return to line 1.

At line 5222 the computer's normal error detecting mode is turned off and the computer is told to go to 5220 instead of stopping with a report code.

The reason for this can be found in the next statement of the same line:

```
IF VAL Z$>=Ø AND VAL Z$<=15 THEN GOTO 5230
```

At this point the computer is looking for a number representing a file line or the number 0. This input goes into a string (Z$) which is then evaluated by the above statement. If the number is between 0 and 15, the program continues to5230. Otherwise 5222 forces the computer back to do the input again.

Whenever the VAL function is used a potentially dangerous situation arises. VAL takes the numbers in a string and evaluates them or works them out as if the string were an arithmetic problem. The danger is that when VAL is used on a string, that string must have numbers in it or the computer doesn't know what to do. The machine could work out situations like:

```
LET Z$="1+1"
PRINT VAL Z$
```

or even:

```
LET X=1
LET Z$="X+X"
PRINT VAL Z$
```

but:

```
LET Z$="BOSTON CREME PIE"
PRINT VAL Z$
```

will stop it cold every time.

In Pro/File 2068, therefore, if you type a number into Z$ at 5220 that represents a file line, an error would not occur. But if you type "BOSTON CREME PIE" instead, the program would stop were it not for the ON ERR command. If an error occurs due to a faulty input, the computer goes back to 5220 to input the number again.

When a number between zero and fifteen is typed, line 5230 is executed. Here the normal error stopping procedure is restored and the Basic variable, S, is assigned the value of Z$, the number you just entered. If this number is zero, the line sends the computer back to 119 to search. Whenever S=0 the program runs through a search displaying files as they're found rather than in any particular order.

If S is not zero, a display of files in an ordered fashion is desired, and S is the line number that is the ordering criterion. Two "ordering strings" are initialized by 5240. P$ is made to equal 5 blank spaces and Q$ is set to 5 lower case Z's (zzzzz). These strings represent the lowest and highest (starting and ending) points for ordering files. Then the computer jumps into the search sequence by going to line 119.

The computer makes this branch into the search routine with two vital preconditions determined: the search command is entered into X$ and the variable, A, equals one. Thus, the data is searched for matches to X$, found files are printed and stored in the E$ array as normal, but since A=1 line 1045 is executed before the Display Options are printed: GOTO 7000.

This is where ordering takes place. Remember that the computer arrives here with:

E$ holding the found file
S pointing to the file line which is to be the basis of the ordering.
or zero if ordering is not required

Line 7000 tests the value for S and if its zero, you go to line 7202 where if Y is also zero (no output to printer), the computer jumps back to line 1050 to print the Display Options.

If Y is not zero printer output is requested and the printer routine starting at 7205 is executed.

That is what happens when no ordering is needed--or when line 7000 tested the value of S and found it to be zero. If S is not zero line 7005 compares the variables B and P. When B=P, you may recall, it means that all data has been scanned: the search is complete. If this is the case you go to line 7070 where the "ordering strings", P$ and Q$ are advanced. More on this in a minute.

If B<>P (the search is not yet complete) line 7010 creates S$. This string becomes the first 5 characters of E$(S) or put differently, the first 5 characters of the found file's line that the ordering is going by.

Then 7020 compares these characters and P$. If they are equal, the computer goes to 7200:

```
POKE PEEK 64026+256*PEEK 64027,0
```

This is one of those strange looking commands that appear very complex but actually perform a very simple function. The clue to understanding this line is to remember that addresses 64026 and 64027, the Pro/File machine code pointer called FADR, stores the address of the start of the file which is currently being displayed on the TV screen. Therefore, when you POKE the PEEK of 64026 and 64027 you are really poking the first character of the file (the asterisk). In this case, you change the asterisk (CHR$ 42) to CHR$ 0, the value that you poked.

Knowing the reason why this is done is much more important than knowing how it is done. The entire ordering procedure at this point has not yet been fully explained, so its understandable if it appears about as clear as mud. Take my word for it, though, that the only time the asterisk is poked to a

CHR$ 0 is when the file found to contain the search command is ready to be displayed or lprinted in the proper order. You will see as this explanation progresses that an ordered search requires that the computer go through your files many many times to find the right file to print at the right time. Once a file is printed, we can safely assume that this file no longer needs to be checked by the computer. It was printed once so it doesn't need to be printed again.

Poking the asterisk to a CHR$ 0, then is like raising a little flag that tells the computer that it already processed this file. Part of the machine code search routine looks for CHR$ 0's and when it finds one it continues on to the next file. If the machine code finds an asterisk, it stops to perform its ordering tasks.

To visualize what goes on here, imagine yourself with a large pile of index cards (files). On each is a persons name along with a list of their hobbies. Your job is to go through every card to find those who build ships in bottles and list the people in alphabetical order. The one restriction you must follow is that you cannot rearrange the index cards. Instead, you scan each card for "ships in bottles" (that's your search command). As you find these hobbyists, you note the names and when you complete your search you write down the name which comes lowest in the alphabet. After you write down the first name you put a little check mark on the card to remind yourself that you already did this card. Then you start again to look for the next ship builder for your list. The check mark saves you time later on. Whenever you come to a card that's checked you know that the name is already on your list. You can skip this card.

Checking off the cards as you do them is the same thing as poking the asterisk into a CHR$ 0. It saves time by eliminating a lot of unnecessary comparison work.

Getting back to Pro/File 2068, after the asterisk is changed, either the file is lprinted or the Display Options are printed on the TV screen depending on the value of Y. This happens only when the found file is determined by the computer to be next in order (when S$=P$ at line 7200).

If S$<>P$, the computer instead proceeds to 7030:

```
IF S$>P$ THEN IF S$<Q$ THEN LET Q$=S$
```

This is the line that determines the next file in order. It is appropriate, now, to take a closer look at the "ordering strings", P$ and Q$, and how they relate to S$, the file characters which are being ordered.

This can be done by stepping through an AUTOSEARCH and watching what happens. For the sake of argument, let's say our data base holds the name/hobby information that was presented earlier. To keep things simple, we'll say the program holds only four "index cards". They consist of the individual's name on the first line and their one and only hobby--ships in bottles--listed on the second line. When printed out they look like this:

```
SMITH, JOE
SHIPS IN BOTTLES
file 1

JENSEN, RABE
SHIPS IN BOTTLES
file 2

ANDREWS, ROBERT
SHIPS IN BOTTLES
file 3

BAKER, MARY
SHIPS IN BOTTLES
file 4
```

As before, the task at hand is to order the names of every file since they all have "ships in bottles" on line two. Since every file has the individual's name on line 1, ordering shall be done on the basis of line 1. Also, we will be content with screen displays only--no print out--so now, on with the show.

Starting at the Main Menu, the word "AUTO" is typed to initialize the AUTOSEARCH. This takes us to lines 5200 to 5240 where at the computer's request you type:

"N" for NO PRINT OUT (makes Y=0)
"SHIPS" for the search command (X$="SHIPS")
"1" for the line number by which files shall be ordered (S=1)

Other important variables set up by these lines include:

A=1 (meaning AUTOSEARCH is ON)
P$=" 5 blank spaces "
Q$="zzzzz"

Following this, the computer goes to line 119 to start searching for "SHIPS". The first file it finds is the one with Joe Smith's name in it. File 1 gets printed on the screen and dumped into the E$ array.

Because A=1, line 1045 jumps the computer to 7000. Here the value of S is checked and since it is not zero, the computer progresses through line 7050.

First, the computer checks to see if the search is complete (line 7005) If it is, B=P and the computer goes to 7070. In our case the search is not complete. Only the first of four files has been found, so line 7010 creates S$ such that it becomes the first 5 characters of E$(S). The variable S is 1 so E$(1, TO 5) or the first 5 letters on line 1 of the found file go into S$. Thus, S$ now equals "SMITH".

Next, line 7020 compares "SMITH" with P$--or " 5 blank spaces ", and if they're the same (which they're not) the computer goes to 7200. For our case, the computer executes line 7030 instead. This line bears reprinting:

IF S$>P$ THEN IF S$<Q$ THEN LET Q$=S$

If you read the Timex 2068 owner's manual on pages 127 and 128, you'll see how a string can be greater than or less than another string. In this case, P$ is all spaces and is the lowest it can go for Pro/File 2068's ordering purposes. Therefore, S$ must be greater than P$.

Similarly, Q$, being "zzzzz" is the highest or last possible string that the program is designed to handle. Thus the second "IF" is also true--S$ must be less than Q$. The action taken when both "IF's" are true is to make Q$ equal to S$. To rephrase line 7030 in English and in our present situation:

> If "SMITH" is greater than " " (true) then if "SMITH" is less than "zzzzz" (also true) then change Q$ so instead of being "zzzzz", it becomes "SMITH".

Now Q$ is "SMITH". Next, the computer goes to 7230 where if A=1 (true), the search is not complete (B<P, also true), and Y$<>"C" (which it isn't), the screen is cleared of the Joe Smith file and a jump sends the computer to line 150: back to continue the search.

The next file the computer finds is Rabe Jensen's file. Again, line 1045 goes back to the lines at 7000. This time S$ becomes "JENSE" and 7030 compares "JENSE" with " " and Q$ which is now "SMITH". As before, both "IF's" are true so the result of line 7030 is that Q$ now becomes "JENSE". Next 7050 jumps to 7230 which jumps to 150 to search some more.

The third time around, the ANDREWS, ROBERT file is found. S$ becomes "ANDRE" at line 7010. Once again, line 7030 changes Q$ so that it now equals "ANDRE", and we go back to search more.

Things are a bit different when we cycle through to 7000 the fourth time. S$ still becomes the first 5 characters of the Mary Baker file (BAKER), a result of 7010, and the computer still arrives at line 7030, but this time,

after finding that S$ is, indeed, greater than 5 blanks, the computer compares S$--"BAKER"--with Q$--"ANDRE"--and finds that S$ is not greater than Q$ as it was in every case before. Therefore, Q$ is not changed this time. It remains "ANDRE".

Finally, the computer goes back to 150 to search more, but all it finds is "SEARCH IS COMPLETE". When this occurs, the variable, B, is equal to P. The computer returns to the lines starting at 7000, but since B=P, line 7005 sends us to to something different at 7070.

Here, we check what's in P$ (5 blanks currently) and if its "zzzzz" we go to 1050. Line 7070 is false this time around so the next line, 7080 is executed instead:

LET P$=Q$: LET Q$="zzzzz": CLS : LET X$=X$: GOTO 119

After this line is executed, P$="ANDRE" and Q$="zzzzz". The computer goes back into the search. However, this time it starts at the beginning of the data again rather than simply pick up where it left off as it did in the previous four searches. The search command is still "SHIPS", A=1 as before.

Thus, the search turns up the first file again, Mr. Smith. Lines 7000 to 7050 work as before. Because 7070 changed Q$ to "zzzzz", it is greater than "SMITH"--the new S$. Therefore Q$ gets changed to "SMITH".

This cycling through lines 7000 to 7050 produces a result identical to the previous time until the computer finds the third file--the one that has ANDREWS, ROBERT on the first line. At this point, P$="ANDRE" and so does S$. When this happens line 7020 branches the computer to 7200.

After all this searching and comparing, Pro/File 2068 finally found the first SHIPS file in alphabetical order. Line 7200 "checks off" the file by changing the asterisk into a CHR$ 0, and because Y=0 (no print out), 7202 sends the computer to print the Display Options at line 1050. The ANDREWS, ROBERT file is printed on the TV screen.

If you press just ENTER of the options, the computer continues its diligent search. The variable, A, remains 1 so the program resumes its cycle through the lines beginning at 7000 after the search produces a found file.

When SEARCH IS COMPLETE turns up, Q$="BAKER", the next file in order. 7070 puts "BAKER" into P$ and the process repeats until the file with BAKER, MARY on line 1 is found. Ms Baker's file will be the next one printed with the Display Options.

Further ordering proceeds every time you press just ENTER of the Display Options. AUTOSEARCHES may still be only about as clear as mud to you

but if you read this section about 6 times and actually try out the example given, watching what happens on the screen when you try an AUTOSEARCH, it will eventually sink in.

The ordering mechanism built into the program is a bit different from other types of sorting algorithms, namely because the data does not get reordered. It is only the display of data which is ordered.

In summary, the ordering routine is a series of searches and comparisons. Every file matching the search command is examined for the lowest line in order. Q$ keeps track of this line (or the first 5 characters of it). At any given time during a search, Q$ will hold the lowest line (alphabetically) yet found. Every time a search is complete, Q$ will hold the lowest line of all the data. The contents of Q$ then gets shifted into P$. Searching continues until the line in question (line S) matches the contents of P$. This signals that the computer has found the correct file in the correct order. The file can then be displayed or printed out. Each file so displayed is "tagged" so the computer will know that it need not compare the file's contents again.

The purpose of this "tagging" is to save time. The more data you're ordering the more time you save. You can see the savings by actually running the example given. When you do, you'll see all matching files displayed very briefly on the TV while comparisons are made.

To print the first file in order, the computer must flash 7 different files on the screen. The next ordered printing, however, requires only 5 comparisons. The third display is accomplished with 3 more checks, and the last file in order can be found with just 2. It is typical with this type of sort, therefore, that finding the first ordered file takes the most amount of time. Each subsequent file requires proportionally less.

To close out an AUTOSEARCH requires a small bit of housekeeping. Since the asterisks which normally preceed every file are changed to a CHR$ 0, the computer must change them back to asterisks. You may recall way back in line 5 the machine code executed by USR 64268. This is the code that does it.

Readers interested in the mechanics of this sort algorithm may find the "simplified" sort listing given in the appendix. The routine is the same as in Pro/File 2068, but all extraneous lines are removed and ordering is done on a simple string of characters rather than a complex array.

PRINTER FUNCTIONS

The two major aspects regarding printer use with Pro/File 2068 are 1, deciding just what portions of a file should go out to a printer, and 2, once you know, actually sending those parts out to it.

The DEFP function available from the Main Menu and also from the Display Options stands for DEFINE PRINTER, or "define what goes to the printer". Whenever you type "DEFP" at the appropriate prompt, the DEFP routine at line 6500 is executed.

This routine does two things. It lets you conveniently define which lines of any given file should be lprinted, and it translates that information to a form which the computer can conveniently execute.

At line 6510 two separate strings are created which store the printer information. C$ stores it so the computer can understand it, and A$ stores it so you can understand it. 6510 lets you input the printer data into A$.

The required syntax when inputting A$ is a number followed by a "/" followed by another number, then another "/" for as many numbers as you wish. The numbers must be between 0 and 15, and you cannot use a "/" as either the first or the last characters. The one exception to the rule is that you can type "ALL" which the computer translates to mean the entire file--all 15 lines.

After you input your printer definition into A$ at 6510, the same program line tacks a "/" onto the end of it and then checks to see if the first character is also a "/". If you do, you go back to 6510 to repeat the input because a "/" at the start can't be interpreted by the computer.

If you entered a legal input, the next line, 6515, checks to see if you typed the word "ALL". If you did, the following commands are executed by the same line:

```
FOR X=1 TO 15
LET C$=C$+CHR$ X
NEXT X
```

These commands create a C$ consisting of CHR$ 1, CHR$ 2, CHR$ 3, CHR$ 4, and so forth all the way up to CHR$ 15. This is the form which the computer understands: a string of characters, each whose code represents a file line to be lprinted. You'll read more about this in a minute.

If you didn't type "ALL", line 6520 is executed instead. It says:

```
LET D=1: FOR X=1 TO LEN A$: IF A$(X)<"/" OR
A$(X)>"9" THEN GOTO 6510
```

This sets up a loop which checks every character of A$. If the character being checked is less than the slash (/), or if it is greater than the digit 9 the program jumps back to force you to input A$ again.

If the character is within these bounds, the next program line:

```
IF X<LEN A$ THEN IF A$(X TO X+1)="//" THEN GOTO 6510
```

makes sure you didn't end the string with a slash. Go to 6510 if you did. If the character being checked makes it past these tests, the computer is ready to translate the character into C$. Line 6530 does this:

```
IF A$(X)="/" THEN LET N=VAL A$(D TO X-1): LET D=X+1:
IF N<16 THEN LET C$=C$+CHR$ N
```

Remember that the variable D was set to 1 by line 6520. Therefore, this command translated from Basic says:

> If the character being checked is a "/" then create a new variable, N, so it equals the value of the digits that preceed the slash. As long as this new variable is less than 16, add a character to C$ whose code represents the value of N.

The role that the variable, D, plays is this. D starts out equal to 1. The only time line 6530 is executed is when the loop finds a "/" in A$. Since previous checks don't allow for a "/" to appear as the first character of A$, it follows that the value of X must be greater than 1 when 6530 is executed. Therefore, the value of A$(D TO X-1) would have to be the value of the A$ characters 1 to that which precedes the slash (either 1 if the number is a single digit or 2 if it is a two digit number).

Remember that A$(X) represents the slash so the statement in 6530 which says LET D=X+1 moves D to point to the first character after the slash in preparation for receiving the next number to put into C$.

The final test is to be sure that the number in A$ is not more than 15, the maximum number of file lines possible in the program. Line 6535 sends the computer back to re-input the number if it is greater than 15.

Once this is done, the program is ready to do the same thing all over again with the next character. Each character that you type into A$ is thus checked. As long as it is valid, the C$ receives the character whose code equals the number you typed. An illegal entry, however, sends you right back to input the string again.

It is interesting to note that this routine does not care how many lines you wish to lprint. It can range from just 1 blank line feed to every line repeated several times.

With this fact in mind, we can now look at the actual lprint part of the program. This starts at line 7205. Since the number of lines you wish to lprint is not limited, we can say that the length of C$ can be just about anything. Each character of C$ represents a line to be lprinted.

When the computer comes to the print out routine, it does so with C$ preset and a file which is currently on the TV screen also loaded into the E$ array.

Therefore, lines 7205 through 7225 set up a FOR.... NEXT.... loop which repeats as many times as there are characters in C$. The loop treats each character of C$ in turn, lprinting the line of E$ which is pointed to by the code of the character in C$.

For example: if C$ consisted of 3 characters whose codes were 1, 5, and 2 respectively, the loop at 7205 would lprint lines 1, 5, and 2 of the E$ array which holds the currently displayed file. The characters of C$ will have codes of either 0 or some number from 1 to 15. When a CHR$ 0 is encountered by the loop, line 7210 causes the printer to skip a line. A code from 1 to 15 found in C$ makes the computer lprint line 1 to 15 of the E$ array. This is done by line 7220

The jump to this routine can occur either from within an AUTOSEARCH where a print out is desired or from a selection of the Display Options. Line 7230 sends the computer back to line 150 if A=1 (during an AUTO-SEARCH). Otherwise, 7240 returns you to the Display Options.

MACHINE CODE IN PRO/FILE 2068

In order to perform all the functions that it does Pro/File 2068 relies on machine code programming to keep things moving quickly. This chapter will discuss just what the code does and how it works. I'll not promise to turn the machine code neophyte into a whizz-kid programmer with this chapter, but I will make an observation or two on what machine code is and how the neophyte might begin to make some sense out of it.

For some reason, there is a barrier to understanding machine code that is very difficult to break down. There's an equally high barrier facing a writer who tries to explain what machine code is.

I suspect that part of this is due to a limitation of the English language. English, being the old and respected language that it is, evolved long before computers ever did. The words needed to describe computers, therefore, simply have not evolved yet.

Thumb through a few pages of any computer magazine and you may dispute this. Our language (or the printing in the magazines, at least) is filled with new words like "software", "byte", "download", or "buffer". This new vocabulary attempts to describe the thoughts of the person who wrote them, but new words (like new tricks and old dogs) are hard for people to learn. "Words" are meaningless "utterances" if they're not understood.

The obstacle to understanding the new computer lingo is that the ideas and thoughts it is intended to convey are just as new as the words themselves. It is only recently that we ever had to consciously think out tasks so that a computer could understand them. The human brain, a most advanced computer (although somewhat slow), can treat the most complex tasks of inputting data (senses), analysing it in the grey matter, and acting appropriately through the muscles as a matter of course or second nature. We never need to think about all the steps necessary to perform even the simplest tasks.

To do the same thing with a "chip" and a "program" requires very careful forethought every step of the way--forethought that was never required before computers came along. Make no mistake about it, this kind of thinking is new and we humans are not very good at it yet.

All programmers, whether they program in Basic or Machine code, must look at the job they want the computer to do and think about how they would do it if they were the computer themselves. The programmer, therefore, must think a great deal about thinking like a computer.

This kind of thinking is quite different from human thought. The computer's sense and action are very restricted. Often the only input sense is the keyboard, while the action it takes to a given input is limited to a TV screen or printer.

Analysis and decision (or "thinking" as it is sometimes called) is carried out in the Z80 microprocessor chip--the grey matter of the computer. It too is restricted to following a single pre-determined set of instructions (the program) which is comprised of a limited vocabulary (commands). What's more, "second nature" does not exist for the computer. Every task must be spelled out step by step.

With all these limitations, you may wonder how the computer with its one track mind and limited vocabulary ever does anything. Actually, the computer's singlemindedness couples with its ability to analyze and decide very quickly to make up for its other shortcomings. A one track mind thinks clearly and is not easily distracted. The computer does not make mistakes unless it is told to make one by an error in its program.

The computer's speed will just blow you away. Consider for a moment a simple counting exercise. How long would it take you to count from 1 to 65000? If you have any sense at all you'd dismiss the idea as absolutely preposterous. You wouldn't even try.

Well, your TS2068 doesn't know the meaning of the word preposterous. It will blindly attack such a job and in about 13 milliseconds, it'll be finished. That's 13 thousandths of a second!

So, what's so special about counting from 1 to 65000 almost 100 times per second? Well, nothing really. But suppose you programmed the computer so that instead of just counting, it matched a series of numbers against a long list. And every time it finds a matching sequence, it prints the series on the TV screen. Now that is just a few steps away from what Pro/File actually does.

A BRAIN MADE OF THIS?

The Z80 brain of your computer is a very complex array of switches that turn off and on in time to the tick of a clock. What makes the computer so fast is the clock's speed: several million ticks per second. The Z80's sole function is to flip switches and determine whether they are on or off. This is hardly the stuff minds are made of, but what makes things interesting is that you the programmer are the one who determines which switches the Z80 turns on or off. You can do some very interesting combinations.

To the Z80 microprocessor, all this flipping and flopping can mean only two things: a switch is either on or it is off. The voltage at the switch is either high or it is low. It is this dichotomy of states--off or on--that is the basis of everything we do with computers. This is the key to computer nirvana.

THE BINARY NUMBER SYSTEM

The human problem of keeping track of which switches are on and which are off arose early on in computer science. People had to find a way to think of all these switches in an orderly fashion so it would be easy to remember just what was what. The model that arose was a number system based on the 2 on/off states of the computer. In this "binary" system (two states) the digits of a number correspond to the state of the switch: 1 if the switch is on, 0 if it is off. Each switch, therefore, becomes a Binary Digit, or "BIT". Bits are arranged in groups of eight, resulting in one eight-bit number called a "BYTE".

Bytes always consist of 8 digits. Each bit can have a possible value of either a 1 or 0 depending on the state of the switch or bit in question. Looking at the byte as a whole, a value can be attributed to it based on the value of each digit.

We can take every possible combination of 8 bits and find that a byte can be configured in one of 256 possible ways. Each arrangement is regarded as being a different value. If all switches in a byte are turned off, the byte would look something like this:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	0	0	0	0	0

or written more compactly: 00000000. The byte has a value of zero. Other combinations and values are:

```
00000001 = 1
00000010 = 2
00000011 = 3
00000100 = 4
00000101 = 5
00000110 = 6
00000111 = 7
00001000 = 8
```

The 1's used in columns slowly move leftward from the rightmost column until:

```
11111000 = 248
11111001 = 249
11111010 = 250
11111011 = 251
11111100 = 252
11111101 = 253
11111110 = 254
11111111 = 255
```

This progression of high bits moving from right to left is the same behavior that our own decimal number system exhibits except that each of our digits

can be one of 10 possible states or figures. We use the digits 0 through 9 in decimal, the computer uses just 0 and 1 in binary.

It is important to remember that this binary number system is just a model we use to describe the action of the switches in the computer. It is a figment of our imaginations. The computer throws a few switches, and we humans attach all kinds of significance to it. What a typically human endeavor.

MEMORY: THE GREAT BYTE STASHER

OK. Now that we have bytes and numbers, how can we use the computer to transform them into something meaningful? This is where "memory" comes in.

You can think of a computer's memory as a large notebook. Each page is numbered so you can easily turn to page 356 to see what's written there. Since the computer deals in byte size numbers, it is a byte that will appear on any given page. Therefore, each page of the notebook (or each location, or each "address" of memory) has two numbers associated with it: the address itself (the page number) and the one byte value found at that address.

When you type the Basic command: PRINT PEEK 25736, a number gets printed on the TV screen which is the value of the byte located at address 25736. This is the same thing as saying: Read the number found on page 25736 of the notebook. Similarly, addresses in memory can be given values to store. In Basic you type POKE 25736, 128 to write a 128 onto page 25736. This action erases what used to be in memory address 25736 and writes a new value of 128 in its place. Each memory address can store one byte; that is, each can store a number ranging from 0 to 255.

One question now arises. If the computer uses bytes which can be from 0 to 255, how is it that the address can go much higher? Wouldn't the addresses similarly range from 0 to 255? The computer handles addresses of much more than 255 by combining two 8-bit bytes into a larger 16 bit "word".

Memory addresses are always referred to as 2-byte numbers while the value held in that address is a single byte number. For the sake of illustration, let's pull two single byte numbers out of a hat and combine them into an address. The numbers I'll choose are 100 and 136. These numbers represent the binary form:

01100100 for the decimal 100
10001000 for the decimal 136

When we put the two together like this:

01100100 10001000

we wind up with a number 16 bits long, the "high half" being 01100100, and the "low half" being 10001000.

If we were to do the same thing with the decimal numbers these binaries represent, it would look like:

100136

That's one hundred thousand one hundred thirty six. You do not add them together arithmatically, but rather combine two 3-digit numbers to make one 6-digit number.

CONVERTING FROM BINARY TO DECIMAL

When we look at the decimal digits 100136, it comes as second nature to recognize the value. Look at the 16 binary digits, though, and you may wonder how in the world anyone can tell just what address it is supposed to represent in our more familiar decimal notation. Actually, the TS2068 itself can calculate the address using the BIN function. Type:

PRINT BIN 0110010010001000

followed by ENTER. The computer tells you what this number is in decimal: 25736.

Another way to arrive at the same answer is to take the high byte (100) and multiply it by the number of possible values it could be (256). Then add the result to the low byte (136). Arithmatically it would be:

(100*256)+136=25736

When two bytes are thus combined to denote an address, the range of possible values can be from:

(0*256)+0 or 0 to
(255*256)+255 or 65535

In other words, the computer's memory can have a total of 65536 addresses (0 to 65535). The notebook has 65536 pages from which or to which we can read or write a one byte value.

A notebook of this type is a very strange one, indeed. Were we to create a similar one which we could physically hold and look at, it would probably be something like a New York City phone book.

Only one number from 0 to 255 would appear on each page. Page numbers would be present, but since two numbers also from 0 to 255 are combined to indicate which page we are looking at, we'd need a calculator to figure out what the "real" decimal page number is for any given page.

HEXADECIMAL TO THE RESCUE

Fortunately, the wise and kind computer designers have given us yet another number system which is very useful in untangling the mess they left us in converting two byte size numbers into one decimal number. This number system is called HEXADECIMAL or HEX for short. Where decimal is a base 10 number system, and binary is base 2, hexadecimal is base 16.

If you dredge up early recollections of your junior high math days, you may remember how a decimal number could be described as being a series of digits in various columns. Each digit or column, starting at the right and moving left was 10 times greater than the previous column number multiplied by the value of the digit itself. The number 517 could thus be described:

	100's column		10's column		1's column
number---	5		1		7
equals ---	(5*100)	+	(1*10)	+	(7*1)

In hex, you can do the same thing except you have 16 values you can place in each digit (in decimal, you have the values 0-9). Hex digits use the numbers 0-9, then the letters A through F such that:

Decimal	= Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

For a complete conversion from decimal to hex, refer to your TS2068 owner's manual (p. 239). There, you'll find decimal numbers in the column labeled "code" and the hex equivalent in the column labeled "hex".

To describe the individual columns of any given hex number, we would say that each column starting at the right and moving left is 16 times the value of the previous column. A four digit hex number would then have:

4096's column	256's column	16's column	1's column
(256*16)	(16*16)	(16*1)	1

A hex number like 40B8 would thus be equal to:

(4096*4)	+	(256*0)	+	(B*16)	+	(8*1)
16384	+	0	+	176	+	8

When you add them up you get 16568 decimal.

By now, you're probably wondering how this could possibly simplify number conversions. All 65536 decimal addresses of a computer's memory can be conveniently represented in four hex digits. The highest hex number you can have if you limit yourself to four digits is FFFF hex. This equals 65535 decimal, the highest possible address in memory.

It follows, then, that if four hex digits can hold a 2-byte number, one byte can be written using just two hex digits. Earlier, the decimal address 25736 was converted into its 2 byte representation. The high byte was 100 and the low byte was 136. These two numbers do equate to the address, but by converting these bytes to hex (using the chart on pages 239-245 of the TS2068 manual), we can do it more conveniently:

100 decimal = 64 hex
136 decimal = 88 hex

The decimal address 25736 is the same as 6488 hex. When hex numbers are used, we can simply combine two double digit hex bytes to arrive at the hex address. We don't need to calculate the address as we do when we use two byte decimal numbers.

A non-programmer may see little benefit in using hex, but anyone who's dabbled at one time or another with machine code will find that using the hex number system greatly eases the burden of handling addresses and other multi-byte numbers.

There is an achilles heel to hex, however. When you write to or read from the TS2068's memory by poking or peeking an address, you must use decimal numbers because that is all our friendly but dumb computer has been taught to accept. Therefore, if you want to use hex numbers, you must convert them to decimal yourself before you peek or poke.

Don't despair. You don't have to do this by hand. It can be easily accomplished by the computer. Here's a program listing that does the trick. Run the program and input a hex number. The computer produces its decimal

equivalent. To flip to the Decimal-to-Hex mode, press just enter. Then you can input a decimal to get hex. The program was written with expansion in mind. The conversions are made in subroutines at lines 1000 and 2000. You could plug these into a monitor or disassembler of your own design.

For hex to decimal conversions, you GOSUB 1000 with a hex number in A$ and ERR, an error flag variable equal to zero. On return, A$ equals your hex number, the decimal equivalent is stored in a variable C. If an error occurs during the conversion process, the ERR variable returns a value of 1 (see line 35).

In the decimal-to-hex mode, you GOSUB 2000 with a decimal number in A$. The subroutine returns as before, with hex in A$ and decimal in C. Errors set ERR to 1.

Program to Convert Hex and Decimal Numbers

```
  10 LET ERR=0: DEF FN A(H$)=COD
E H$-48-7*(H$>"@"): LET H$="0123
456789ABCDEF"
  20 INPUT "HEX into DEC.?";A$:
IF A$="" THEN GO TO 100
  30 GO SUB 1000: REM [illegible]
  35 IF ERR=1 THEN PRINT "INVALI
D NUMBER": GO TO 1
  40 PRINT A$;" hex=";C;" dec."
  50 GO TO 20
 100 INPUT "[illegible]";A$: I
F A$="" THEN GO TO 10
 110 GO SUB 2000: REM [illegible]
 120 GO TO 35
1000 LET C=0: FOR X=1 TO LEN A$:
 IF CODE A$(X)>=48 AND CODE A$(X
)<=70 THEN LET C=C+FN A(A$(LEN A
$-X+1))*16↑(X-1): NEXT X: RETURN

1010 LET ERR=1: RETURN
2000 FOR X=1 TO LEN A$: IF CODE
A$(X)>47 AND CODE A$(X)<58 THEN
NEXT X: GO TO 2010
2005 GO TO 1010
2010 LET C=VAL A$: IF C>65535 TH
EN GO TO 1010
2015 RANDOMIZE C: IF C=0 THEN RE
TURN
2020 LET P=23671: LET A$=""
2030 LET N=PEEK P: LET N$=H$(N/1
6+.5): LET N=N-INT (N/16)*16+.5:
 LET N$=N$+H$(N): LET A$=A$+N$:
IF P=23670 THEN RETURN
2040 LET P=P-1: GO TO 2030
```

By now, you have waded deep enough into the bowels of computer numbers and organization of memory to experience the thrill of enlightenment. Well... maybe not quite. Actually, don't feel too bad if none of it has sunk in because abstractions such as these rarely fall into place until you get some real machine coding experience under your belt. Read on. The fun is only just beginning.

MEMORY: WHAT YOU CAN DO WITH IT

If you think of a computer as a large notebook of memory addresses, you can think of the Z80 microprocessor chip as a tool that lets you manipulate or process the 65535 bytes at your disposal. Machine code programming is the way you use the Z80.

A computer's memory can be used for two interrelated purposes: for storing data (numbers, letters, etc.), and for storing a program (instructions for manipulating data). In both cases, the data or program is encoded into numbers from 0 to 255 which can then be placed into bytes.

You may recall the childhood game of sending secret messages by substituting numbers for letters of the message: A=1, B=2, C=3, and so on. The computer's encoding scheme is similar. If you refer again to the table in the TS2068 manual called "The Character Set" (p. 239), you will find the "secret" code used by the computer.

Column 1, labeled "code" is the number which represents the character shown in column 2. The first code numbers don't equate with any printable characters, but those from number 32 on up to 255 each represent a character or token (a spelled out command word like "LET").

If you poke an address with one of these numbers, say:

```
POKE 65535, 33
```

you can read the character you just poked.

```
PRINT CHR$ PEEK 65535
```

will print the character representing the number you just poked into 65535. The next code number, 34, if poked in at address 65535 will produce the quotation mark when you execute PRINT CHR$ PEEK 65535 again.

This is a very easy way to store data. A word or group of words could be placed in a consecutive block of memory addresses. Each address stores one character. You could store the word "HELLO" in memory by poking the code for each letter:

```
POKE 65530, 72    (the code for "H")
POKE 65531, 69        ( "E" )
POKE 65532, 76        ( "L" )
POKE 65533, 76        ( "L" )
POKE 65534, 79        ( "O" )
```

Poking and Peeking are used here to illustrate that only one character can be handled at a time. Other Basic commands like LET A$="HELLO" also put the word in memory, seemingly all at once. Basic, however, simply implements machine code located in the ROM operating system to write "HELLO" in the same way: character by character.

If you look at column 4 of the "Character Set" table, labeled Z80 ASSEMBLER, you'll see that each code number can represent something else besides characters. The funny looking words listed in this column are the mnenomic form of Z80 machine code commands.

Whole books have been written on the subject of what these commands do. If you want to learn machine code, you'll need to find some of these books to discover how to implement the various commands. To get you started, refer again to the code 33, the exclamation point. In column 4 we see the cryptic:

```
LD HL, NN
```

This is pronounced "LOAD HL, NN" or "Load the HL register with number NN". A convenient way to think of registers is to think of cash registers with money trays that hold just two different types of bills. Each of the two "slots" can hold one byte's worth (or a maximum of 255) bills.

There are several such registers in the Z80 chip including HL, DE, BC, AF, SP, PC, IX, IY and a large set of "alternate" registers of the same name.

LD HL, NN therefore means load the cash register called HL with a number. It's not apparent from this discussion, but you can take my word for it that the actual value placed in HL is taken from the two bytes of memory which follow the command itself. Thus if you poke:

```
POKE 65530, 33
POKE 65531, 1
POKE 65532, 0
```

and then execute the machine code instructions, HL would be loaded with the number 1. Notice two things about this instruction. First, whenever the computer comes to execute a 33 (or LOAD HL,NN), the next two bytes after the command (the 33) are always used to supply the value that actually gets loaded, even if the number is small and can be written using just one byte.

Second, the order in which the numbers are loaded are the reverse of what you might expect. By unalterable definition, the first tray (H) gets loaded with the second byte of the number (address 65532) and the second tray (L) gets the first byte (address 65531).

You get the computer to execute machine code by implementing the USR function in Basic. This function is always followed by a number which represents the address where you wish to begin executing machine code. A Basic statement like:

```
LET X=USR 65530
```

tells the computer to break from Basic and begin executing machine code beginning at address 65530. The computer effectively performs a GOSUB to the machine code. It will not go back to Basic until it comes to a RET

(short for RETURN) instruction with the code 201. It then resumes where it left off. When the computer makes this return, it does so with a value of whatever happened to be in the BC register when the RET instruction was executed.

Therefore, when you say:

```
LET X=USR 65530
```

X will equal the value of BC when the machine code returns. From a programmer's standpoint, this number can be put to good use by Basic. You can use the USR function with any Basic command that requires a number. For example:

```
PRINT USR nnnnn
LET X=X+ USR nnnnn
GOTO USR nnnnn
IF USR nnnnn <> USR xxxxx THEN.....
```

In each case, the machine code starting at the specified address is executed and on return to Basic the number found in the BC register is evaluated and used to complete the execution of the Basic command.

HOW TO WRITE MACHINE CODE

Up until now, only crude illustrations of machine language have been given. Now I'm going to show you how to do something useful with it: how to add one plus one and get the correct answer. Furthermore, I'll show you how to do it several different ways.

First, reset your computer by turning it off and then back on. Next, you must lower RAMTOP to reserve a block of memory that won't be altered by the Basic operating system. Enter:

```
CLEAR 65000
```

to do this. The effect of this command is to make the address 65000 the last address used by Basic. Everything from 65001 to 65535 is still there, but the Basic system will not use it. Now poke these numbers into the following addresses:

```
POKE 65001, 33
POKE 65002, 1
POKE 65003, 0
POKE 65004, 1
POKE 65005, 1
POKE 65006, 0
POKE 65007, 9
POKE 65008, 68
POKE 65009, 77
POKE 65010, 201
```

What you just poked into memory was a 10 byte machine code program that adds 1+1. Looking at the numbers (33, 1, 0, 1, 1, etc.) reveals little about how the program actually works. However, if you go to the "Character Set" table and translate the numbers into the Z80 assembly language you can begin to make some sense out of the codes.

The procedure for doing this is often termed "Disassembling" a machine code program. The reverse--converting the mnenomics into numbers which the computer understands is called "Assembling" a machine code program.

You can buy programs which do all this dirty work for you (a very good one is listed in the Bibliography), but beginners would do well to "cut their teeth" on machine code by doing it by hand at least for a while.

HOW TO UNDERSTAND MACHINE CODE DISASSEMBLIES

Regardless of how you do your disassembling (or assembling)--by hand or by computer--you will inevitably find yourself working almost exclusively with disassembled program listings of machine code mnenomics.

You probably have seen these published in magazines with little or no explanation of what they are or how you work from them to encode them into numbers acceptable to your computer. A disassembled listing of the program you just poked in looks like this:

Address	Code	Mnenomic	Comments
65001	33	LD HL,0001	Put the next two bytes into HL
65002	01		
65003	00		
65004	01	LD BC,0001	Put the next two bytes into BC
65005	01		
65006	00		
65007	09	ADD HL,BC	Add the two registers leaving result in HL
65008	68	LD B,H	Transfer value in H to B register
65009	77	LD C,L	Transfer value in L to C register
65010	201	RET	Return to Basic with BC holding the result of the addition

Disassemblies can be exclusively in decimal like this one, they can be in hex, or they can list both. What they will all tell you is the address where each byte is located, what the value is for each byte, the mnenomic form of the instruction, and usually a few extra comments to clarify points in the program. Therefore, if you had a machine code listing and you wanted

to enter it into your computer, you would poke the value taken from the column titled "Code" into the memory address indicated in the first column. Some machine code listings (like those given for Pro/File 2068's routines) are done completely in hexadecimal. The codes and addresses for these must be translated into their decimal equivalents before you poke them in.

In any event, the columns labeled "Mnenomic" and "Comments" are there only to help you understand what the code is supposed to be doing. The computer does not use them in any way. For "poking purposes" they can be ignored

Now that we have layed out the disassembly of "1+1", it can be scrutinized. The program is simplicity personified. Starting at address 65001, the HL register pair is assigned the value of 1. Then BC is similarly given an equal value. The instruction at address 65007 adds the two register pairs together and stores the sum back in HL. The next two instructions, LD B,H and LD C,L take the sum in HL and transfers it--one register at a time into BC. Thus, when the computer returns to Basic, BC holds the answer. When you type:

```
PRINT USR 65001
```

the answer, 2, is printed on the TV screen.

Just like in other languages, there are countless ways to say the same thing. For example:

Address	Code	Mnenomic	Comments
65001	01	LD BC,0001	Start with value of 1
65002	01		
65002	00		
65003	03	INC BC	"Increment" or "Increase" the the value in BC by 1
65004	201	RET	Return to Basic with BC holding the new value

After you poke these new values into addresses starting at 65001, PRINT USR 65001 will, as before, print a "2" on the TV.

Here is a slightly different way to perform the addition:

65001	33	LD HL,65011	Make HL point to address 65011
65002	243		
65003	253		
65004	62	LD A,01	Load Accumulator with next byte
65005	01		
65006	14	LD C,01	Load C-register with next byte
65007	01		
65008	129	ADD A,C	Add them together
65009	119	LD (HL),A	Put result in address specified by H
65010	201	RET	Back to Basic

In this example, instead of returning to Basic with the sum in BC, the machine code loads a memory location with the answer. First, HL is given the value of 65011 which is the address of the first byte after the RET instruction. Then the A and C registers are each given the value of 1 and are added together. The result is stored in the A register, otherwise known as the Accumulator. Finally at address 65009, the instruction LD (HL),A loads the memory address pointed to by the HL register pair with the value found in A.

After you return to Basic you can PRINT PEEK 65011 and find the answer sitting there.

Sticking numbers in memory locations like this can be very useful in many other ways. Here is a routine that takes the word "HELLO" from one block of memory and moves it onto the TV screen:

Address	Code	Mnenomics	Comments
65001	33	LD HL,65014	HL points to start address
65002	246		of "word"
65003	253		
65004	126	LD A,(HL)	Put character in A
65005	215	RST 16	Print it on TV screen
65006	35	INC HL	Point to next character
65007	62	LD A,00	Reset A to zero
65008	00		
65009	190	CP (HL)	Compare character in (HL) with the zero that's in Acc.
65010	194	JP NZ,65004	If they're not the same then
65011	236		go back to 65004
65012	253		
65013	201	RET	Otherwise, back to Basic

After you poke addresses 65001 to 65013 with the values above, poke the addresses 65014 and above with codes for the letters H, E, L, L, and O. These would equate to:

```
POKE 65014, 72
POKE 65015, 69
POKE 65016, 76
POKE 65017, 76
POKE 65018, 79
```

When you execute the machine code using the Basic command:

```
PRINT " " AND USR 65001
```

the routine will put the word "HELLO" on the TV. In fact, you can poke any word or sentence and have it printed by the above routine. A couple of interesting things are happening in this machine code.

First, RST 16 (Restart 16) is one of several restart commands which cause the computer to "GOSUB" to the specified address. In this case address 16 decimal is called. This is the ROM subroutine responsible for sending characters out to the current output channel like the TV or printer.

If you place the code for the character you wish to print in the Accumulator and then use the RST 16 command, that character will be output and the computer returns to where it started.

Second, CP and JPNZ come from a very useful family of commands which compare (CP) bytes or test for certain conditions and then jump (JP) accordingly to other portions of the machine code. This is roughly the same as the Basic rendition of IF.... THEN GOTO....

In the example, after the character found at the address pointed to by HL is printed using RST 16, HL is incremented to the address where the next character can be found. Then the Accumulator is loaded with zero and the CP (HL) instruction compares the value in the Accumulator with that found in the address pointed to by HL.

The Z80 does this by subtracting one value from the other. If both hold the same value, the result would be zero. If they're not the same, the subtraction would result in some other number.

Any time arithmetic is performed by the Z80, a special flags register has bits automatically raised or lowered depending on the outcome. One flag, the Zero flag, goes high any time a subtraction results in zero. The JPNZ command tests this zero bit. It jumps if it finds the bit low (NZ=Not Zero), an occurrence when the two values being compared are not the same.

If the Zero flag is high, the Z80 ignores the command and executes the next one in order. The outcome is that as long as the computer finds non zero codes in the addresses specified by HL, it is not finished printing the message yet, and the program loops back on itself. As soon as a zero is found (the first unpoked byte after the message), the machine code returns to Basic.

Assuming you've read and tried these exercises, and you have at least a vague idea of how they work, you now probably know just enough to be dangerous. You still have a long way to go. These examples were not intended to make you fluent in all the possible Z80 instructions, but rather attempt to illustrate that the instructions available to you are there to aid you in managing your 64K bytes of memory. The Z80 gives you over 600 commands that test and compare, shuffle, perform arithmetic, manipulate individual bits, input bytes from other devices, and output them in the reverse direction. By using this "vocabulary" with tact and creativity you can make your computer move the world.

HOW TO MODIFY PRO/FILE 2068

No program is going to fit all the needs of all the people who use it. Unlike other tools though, a computer program can be very easily changed to provide different features for different requirements. In fact, this flexibility, the ability to change a program's function as needs change, is for me, one of the computer's most attractive features. Programs (read that "tools") become evolving processes. Nothing is fixed in granite. If you don't like something, rip it out and plug in something that works better. As new techniques are discovered to accomplish a computer task faster and more efficiently, they can simply replace the older, out-moded procedures.

There is just one hitch: If you want to get inside your computer and make it do what you want, you've got to learn how to communicate. You must know at least the fundamentals of programming. A current idea in the computer world would have us believe that most people do not want to know how to program. Instead, they want ready made programs that they can treat like automobiles, something that lets them drive off at the turn of a key without thinking about what's rumbling under the hood.

People that hold this opinion are turning their backs on 95% of the true power their computers have to offer them. Armed with a little programming know-how, you can wield enormous influence over your computer. It makes the difference between the computer shaping your behavior or you shaping the computer's behavior.

The best way to learn how to program is to read everything you can find about programming and spend a lot of time experimenting at the keyboard. Start by entering a simple program listing. Watch what happens when you run it. Change the program by plugging in different numbers or additional program lines. Observe the results. Even mistakes will teach you something about how your computer works.

This chapter will show you how you can change PRO/FILE 2068. Several examples are given to get you started on ideas of your own. It is assumed that you already know how to enter and edit program lines. You should also be familiar with the TS2068 keyboard. If you are an absolute rank amateur in these areas, read the TS2068 owner's manual. You may have to hunt for the information you need, but you'll find it eventually.

THOUGHTS ON MEMORY CONSUMPTION

When you break into the program listing by typing SHIFT 1, STOP, ENTER after the Main Menu is on the screen, you can type PRINT FREE and ENTER

to find out how much memory remains that is unused. A "stock" PRO/FILE which is loaded for the first time will have 1505 free bytes of memory. As you add, delete, or alter program lines, the amount of free memory will change.

THE MORE PROGRAM LINES YOU ADD, THE LESS FREE SPACE WILL REMAIN

Generally, small modifications will take up little or no extra memory, but major revisions could get you into trouble if you need to use more bytes than what remains free.

For the programmer who wants to add 2000 bytes more of program modifications where only 1505 free bytes remain, it may seem like an impasse has been reached. There is a simple way around this apparent restriction. That is, file storage capacity can be reduced to yield more free bytes.

To reduce file storage space, change line 9996 so that instead of dimensioning D$ to 28020 characters, it is some number smaller. For example, to free up another 1000 bytes of memory for additional programming, change the DIM D$ statement in 9996 to:

```
DIM D$(27020)
```

After you do this, type RUN 9996. This will reinitialize everything. You will have 1000 less bytes of data storage capacity, but upon typing PRINT FREE, you'll discover that you have 2505 bytes--1000 more than what you had originally. Whenever you begin to run short of memory, remember this:

TO GAIN MORE ROOM FOR PROGRAMMING, DECREASE THE SIZE OF THE D$ ARRAY

Whenever you decide to open up more memory in this way remember that there is a trade-off to be made. Additional features can be added only at the expense of reduced file capacity. If your additional programming is written efficiently, your new ability to access what data you have left is usually worth the investment.

Important: As noted above, you should ALWAYS use the RUN 9996 command to reinitialize PRO/FILE 2068. For reasons that I do not fully understand at this time, using GO TO 9996 will foul up the program's operation. This can be illustrated by trying it for yourself. Load the program, break into the listing, type GO TO 9996. After you work your way to the Main Menu, try making a search or try adding a new file. ALWAYS USE RUN 9996 TO REINITIALIZE THE PROGRAM.

MAKING YOUR CHANGES PERMANENT

Once you are satisfied with the modification, your change will be made permanent by saving it on tape. This is easily accomplished by simply operating the program and typing SAVE from the Main Menu. All changes made to PRO/FILE 2068's BASIC as well as any updates to your files will be saved. When you reload the tape, your modified program will go into the computer right along with your data.

HOW TO MAKE BACK-UP COPIES OF PRO/FILE 2068

You should make several back-ups of PRO/FILE. You never know when something might go wrong. Little fingers could spread peanut butter all over a tape, your recorder could suddenly decide to eat your cassette, you might accidently record over your one and only copy, or you might make six copies only to discover later that when you made them, the cables were plugged in backwards. All these things sound silly, but every one of them has happened to me. Take it from somebody who knows.

The PRO/FILE 2068 master tape has 3 different programs which get loaded into the computer. First, a simple "loader" program named "pro/file" goes in which lowers ramtop. This program prints the initial display of my name and copyright notice along with the flashing "LOADING... Please wait" at the bottom of the screen. The "loader" also loads the two other programs that follow it on the tape: the machine code bytes titled "p/f"CODE 63488, 2046 and finally the Pro/File Basic which is also called "pro/file". The Basic was saved so that it would start running automatically at line 9996.

To make your own back-ups, you must put the same programs on your cassette. Here is a step-by-step procedure.

1. Power up the computer and enter this short listing. This is the "loader" program described above.

```
 1 BORDER 0
 2 PAPER 0
10 CLEAR 63487
20 LOAD "p/f"CODE 63488,2046
50 LOAD "pro/file"
```

2. Insert a fresh cassette into your recorder and enter the command:

```
SAVE "pro/file" LINE 1
```

3. After the loader is saved, do not rewind the tape. Remove the cassette and load the PRO/FILE master tape.

4. When the "Press C-to Create, L-to Load prompt comes on the screen go to the Main Menu by entering the "C" option.

5. Now break into the program listing by typing SHIFT 1, STOP, ENTER.

6. Replace the cassette you just made and get ready to save the machine language onto the tape so that it immediately follows your loader program.

7. Enter the command: SAVE "p/f"CODE 63488,2046
   to make the save.

8. Stop the recorder when the save is finished. Then type:
   CLEAR and ENTER. Then: SAVE "pro/file" LINE 9996
   Start the recorder again and save a copy of PRO/FILE's Basic.

Now you have a second tape of your PRO/FILE 2068. To reload it, type the command: LOAD "pro/file". The loader program will take over and further load the machine code and Basic.

Making back-ups of the master tape is insurance that you will always have a spare copy should disaster strike. There is another benefit too. It is easier to load tapes that were made by the same equipment that made them. If you find the master tape balky to load, use one of your back-ups to get the program into the computer instead. You will find it much more reliable than the original.

HOW TO ADD A VERIFY FEATURE TO PRO/FILE 2068

Add lines 60 and 8000, and alter 107 so that they look like the listing below to implement a tape verify function. This routine is executed when you type SAVE from the Main Menu. After the program saves, the prompt in line 107 asks you to rewind and play the tape to verify it. If you decide not to go through the verify procedure, you can press BREAK to stop. The routine ends by placing the Main Menu on the TV screen. If a tape error occured or if you press BREAK, line 60 flashes "TAPE ERROR" in the display. If the verification checks out ok, this signal does not get printed.

```
  60 ON ERR RESET : IF X$="TAPE
ERROR" THEN PRINT AT 20,0; FLASH
 1;X$
 107 IF X$="SAVE" THEN ON ERR GO
 TO 8000: SAVE F$ LINE 1: PRINT
AT 20 0;"REWIND TO VERIFY/BREAK
TO STOP": VERIFY F$: GO TO 1
8000 LET X$="TAPE ERROR": GO TO
4
```

HOW TO CHANGE A FILE NAME

As the program stands, the only time you can create a new file name is when you create a new empty file by pressing "C-To Create" after you load the master tape. If you ever wished that you could change the name you gave your data base after the initial name was assigned, this modification is for you. Add this one program line:

```
101 IF X$="NN" THEN GO TO 9998
```

Now, any time you feel the need, type "NN" which stands for "New Name" instead of a search command at the Main Menu. Line 101 sends the computer to line 9998 which is the same line that asked you to input a file name originally.

This routine works only if you enter "NN" from the Main Menu. If you try to do it from the Display Option Menu, a normal search for "NN" will occur.

HOW TO "RESTART" OR CLEAR A FILE OF DATA

Another very simple command that can be added is a RST or "Restart" command. Adding line 104 below will allow you to erase all information from your files without changing anything else such as file name, print format, etc. This line has the effect of wiping out all your records, allowing you to "Restart" with a clean slate.

```
 104 IF X$="RST" THEN LET P=20:
GO TO 1
```

Note that I said it has the "effect" of wiping out your data. Nothing is ever really erased. Line 104 simply resets the Basic variable P to its original value of 20. The data is still stored in D$, but the computer doesn't know to look for it. After a RST, new data added will slowly overwrite the old.

ADD A "TICK" FOR AUDIBLE FEEDBACK WHEN YOU ADD DATA

Some kind of audio feedback is very helpful when you add or edit files. Add this line to Pro/File 2068 to give you a "tick" every time you press a key. When you can hear data going onto the screen, you don't need to constantly watch the screen to make sure you hit the key correctly.

```
5050 BEEP .001,50
```

SAVE PRINTER PAPER BY NOT LPRINTING BLANK LINES

If you have files of varying lengths, lprinting the shorter ones can cause you to waste paper because printer output is defined by the DEFP function rather than the length of the file in question. Some files may use all 15 for text while others might consist of just one or two lines. If you want to print out every file, and you want to be sure that every line gets printed, you would define the printer with the ALL command. This prints every line of the large file, and the same for a short one. The problem is that you also get a lot of blank lines with the short file because "ALL" prints the entire screen display--even if it is only blank lines.

This alteration makes the computer ignore blank spaces. Lines of text will be printed as usual, but blanks between lines or at the end of the TV display will not appear on paper unless you specify that a space be printed by inserting zeros in your DEFP command.

Only one program line needs to be changed:

```
7220 IF CODE C$(X) AND E$(X)>"
THEN LPRINT E$(CODE C$(X))
```

There are 32 blank spaces between the quotation marks.

ALPHABETIZE MORE THAN THE FIRST FIVE CHARACTERS OF A LINE

Noted elsewhere is the fact that when you ask for an alphabetized search, files are ordered by the first five characters of the line specified. For most jobs, this is sufficient. Occasionally, however, you might wish to order a group of last names that are the same with only the first initial which varies.

If the names are more than five characters long like:

Peterson C.
Peterson A.
Peterson L.

Pro/File would be satisfied with the order the names appear because it doesn't give a hoot about anything but the first five letters--PETER ....

Fortunately it is fairly simple to change Pro/File 2068 to make it check more than the first five characters. Alter these four existing program lines so that P$, Q$, and S$ are ten characters long rather than the five that they are. Doing so will cause Pro/File to alphabetize by the first ten letters of a line.

```
5240 LET P$="          ": LET Q$
="zzzzzzzzzz": LET X$=X$: GO TO
119

7010 LET S$=E$(S TO 10)

7070 IF P$="zzzzzzzzzz" THEN GO
TO 1050

7080 LET P$=Q$: LET Q$="zzzzzzzz
zz": GO TO 7070: LET X$=x$: GO TO 119
```

DISABLE AUTO-REPEAT IN ADD/EDIT MODE

If you find you have a heavy touch on the keyboard and every time you press a key, two or three letters get printed on the TV screen, this modification will make things a bit easier for you. Alter lines 5040 and 5103 so they look like the lines below. When you type with the program modified in this way, the computer waits until you take your finger off a key before it accepts a new character. The special function keys (Caps lock, Delete, Insert, Arrows, etc.) still repeat automatically so you can move the cursor across the screen using the arrow keys simply by holding down the desired key, but letters to be printed on the screen go on just one character at a time.

```
5040 PRINT INK 6+i;AT l,c; FLASH
 1;SCREEN$ (l,c)

5103 GO TO 5010+(93*(CODE INKEY$
,13));
```

GO DIRECTLY TO LPRINT FROM THE EDIT MODE WITHOUT GOING TO THE MAIN MENU

This enhancement lets you print out a record while you're editing it without the need to close the file and go to the Main Menu first. Whenever you press "THEN" (symbol-shift-G) the computer branches to the LPRINT section of Pro/File. It prints the file out, then returns to the ADD/EDIT mode where you can continue editing your data. Printing occurs as defined by the current print format.

To include this feature, add these new program lines:

```
5027 IF Y$=" THEN " THEN GO TO 7
205

7227 IF Y$=" THEN " THEN GO TO 5
005
```

GO FROM "EDIT" TO NEXT ENTRY WITHOUT GOING TO THE MAIN MENU

If you have a group of files which you update regularly, this addition lets you search for the first one, edit it, and then go to the next file matching the same search command immediately upon closing from the ADD/EDIT mode. Add line 10 and alter line 5016 as follows:

```
  10 IF Y$=" STEP " THEN LET X$=
X$: GO TO 119

5016 IF INKEY$=" STEP " OR INKEY
$=" STOP " THEN LET Y$=INKEY$: G
O TO ERR RESET: FLASH 0: GO TO 500
0
```

This routine, like the previous one, adds a new command to the ADD/EDIT mode. It creates a second way to close a file. Press "STOP" (symbol-shift-A) as usual to close and jump to the Main Menu. Press "STEP" (symbol-shift-D) to close and jump to the next file display that matches the search command.

AUTOSEARCH IMPROVEMENT IS EASY ON THE EYES

Add, alter, or delete Pro/File's Basic as noted in the following list to eliminate the rapid flashing of files while in the Autosearch mode. The screen goes dark while the computer hunts, and shows you only the correct display. Besides the cosmetic effect, this routine trims a few seconds off the search time, and allows for editing or deleting files without losing track of where it is in the ordered output (provided you add the "EDIT to NEXT ENTRY" modification given previously).

```
   3 LET A=0: LET X$=""
   4 RANDOMIZE P: POKE 64035,PEE
K 23570: POKE 64036,PEEK 23671:
RANDOMIZE USR 64268
   5 BORDER 0: PAPER 0: LET A=0
 DIM E$(15,32): CLS
   8 IF PEEK 63611=0 AND Y$<>"D"
 AND Y$<>" STEP " THEN GO SUB 73
00
  10 IF Y$="D" OR Y$=" STEP " TH
EN LET Y$=X$: GO TO 119
104 PRINT AT 21,0;X$: IF A THEN
 GO SUB 7300: GO SUB 7310: GO SU
B 7300
4000 LET P=P-USR 63640: IF Y$="D
" THEN LET Y$=CHR$ (CODE "D"+A):
 GO TO 4
5200 GO SUB 7300: LET a=1: PRINT
 AT 14,0: PAPER 0: INK 7: PRINT
OUT "  (Y/N)";;;;;;;;: INPUT Y$: L
ET Y=Y$="Y"
6000 LET D$(P)=CHR$ (42-A*42): F
OR X=1 TO 15
6030 GO TO 4-(Y$=" STOP ")+(A*(Y
$=" STEP "))
7070 IF P$="zzzzz" THEN GO TO 10
47
7202 IF Y=0 THEN GO TO 1047
7240 GO TO 1047
7300 LET V=PEEK 63611: POKE 6361
1,215-V: POKE 63597,215-V: RETUR
N
7310 PRINT AT 0,0;: LET E$(1)=""
   IF USR 63575 THEN
7311 RETURN
```

The last two modifications, "Autosearch Improvement" and "Edit to Next", lay the groundwork for "Programmable Search Commands" alluded to in the chapter explaining how Pro/File 2068's Basic works. As promised, here is a description of what these commands are and how you can use them.

PROGRAMMABLE SEARCH COMMANDS DO MORE THAN JUST PRINT DATA

I use the term "Programmable Search Command" to describe a method whereby you can implement a search in the normal way, but along with that search, you can apply a special action to the data found by the search.

Built in to the program are two such actions: printing on the TV screen and lprinting on the printer. In other words, you input a search command, the computer finds the particular data you want, and then the action it applies to that data takes the form of a TV display or a printout.

Imagine what you could do if other actions could be carried out. Some specific ideas are block manipulations of data stored in the program, counting files found by a search, applying math functions to numbers found in an individual record, or sending data out to a modem or floppy disk rather than a printer.

The key to capitalizing on this potential lies in line 120 of Pro/File's Basic. This, you may recall, is where the search command is placed into a special buffer or holder before a search is made. Line 120 uses machine code to do this and it is implemented by means of the Basic RANDOMIZE USR command. This command sets a Timex system variable called SEED to a number. Just which number is determined by the machine code that loads the search buffer.

When you type a regular single or multi-word command, the SEED system variable (address 23670) will always hold a 1. If you enter a command so that it ends with the reverse slash (on the "D" key) followed by some other single keypress, SEED will hold the code for the last key you typed. The search will be for every character up to the reverse slash, but SEED will hold a number that you could use as a flag to jump out of the program and into your own modifications.

The routines that follow demonstrate a few possibilities in using this flag as well as show the power in Pro/File's ability to do something other than just print.

BLOCK DELETE
PURGES SPECIFIED FILES

You say you have a mailing list in Pro/File 2068 and you want to delete every person in a certain category? This routine makes it simple. All you do is type as a search command the word found in each file you wish to delete, ending it with a reverse slash and a "D" (for DELETE). The computer wipes out every file that holds a match to the word you type in. When it's done, the "SEARCH IS COMPLETE" display is printed.

This modification would prove to be useful in a case where several different kinds of files are stored in the same program. As more and more data accumulates, and you find yourself running out of space, you can block delete one kind of file and leave the rest.

For example, if you have both your stamp and your coin collection in one copy of the program, you could "split" the collections by block deleting your stamps. After doing so, rename and save the updated program. Then, reload the old copy that had both collections together and block delete the coins. Once you rename and resave this program, you'll have two programs: one of stamps and one of coins.

To add this feature you must have the "EDIT to NEXT ENTRY" modification as well as the "AUTOSEARCH IMPROVEMENT" just shown already in place and functioning properly. Then add these program lines:

```
 130 LET J$=CHR$ PEEK 23670
 200 IF J$="D" THEN LET A=1: LET
 S=0: LET Y$="D": LET Y=0
1049 IF E$(1, TO 18)<>D$(2 TO 19
) AND CODE J$>32 THEN GO TO 8500
+CODE J$
8558 GO TO 1055
```

BLOCK SORT
REARRANGES DATA

One curious aspect of Pro/File 2068's alphabetizing routine is that when it is carried out, data does not actually move. It is only the DISPLAY of information that's affected. This routine really does move your files around.

An advantage to doing this is that printing out data is much faster when you call for a simple "dump" to the printer rather than having it sorted in some special way. If you have a list of files, you could, at your leisure, block sort your data, save it on tape, and then when you need a fast print out, use the suitably arranged copy of the program to make the hard copy.

The BLOCK SORT works in the same way as the previous Block Delete function. Type a search command followed by a reverse slash and a number from 1 to 9. Ordering will be done on the basis of the line number you type. Only those files holding a match to your search command will be ordered. Use an asterisk as a search command to order every file.

Add Block Sort capabilities by first adding the "EDIT to NEXT..." and the "AUTOSEARCH IMPROVEMENT" routines as shown. Then add these lines:

```
 130 LET J=CHR$ PEEK 23670
 140 IF A=0 THEN IF CODE J$:48 A
ND CODE J$<58 THEN LET S=VAL J$:
 LET A=1: LET Y=0: LET Y$=" STEP
 ": GO SUB 7300: LET P$="      "
 LET 0$="zzzzz"
8548 GO TO 1
8557 LET P=P-USR 63640: GO TO 60
00
8567 GO TO 1
```

PERFORM MATH FUNCTIONS AND TABULATION ON DATA STORED IN FILES

You can transform Pro/File the file manager into Pro/File the accounts receivable by adopting this enhancement. It is somewhat on the long side with many different steps, but with it, you have two new functions that offer enormous potential. They are 1) File counting (or tabulation) which tells you HOW MANY files are found to match a given search command, and 2) math capabilities on up to two different numbers stored in each file.

Tabulation gives a simple form of data analysis. If you use the program to store research data, you can use the counting function to tell you how many files you have that fit a certain criteria. In business, you could use it to tell you how many customers purchased a certain product or how much you consume a given material.

The math function can be used to tally quantities or amounts associated with files. The program can handle one or two different numbers per file. It keeps a running tally throughout a specified search. This routine is also capable of handling simple arithmatic expressions besides just a number. You can refer to a number as "25" or "10+10+10-5". Fractions, decimal points, plus, minus, and divide are all catered for.

If you're wondering why multiplication was not mentioned, it is because of a peculiar situation that arises when you add this mod. The multiplication sign (*) cannot be used in a file because it also serves as a marker at the beginning of each individual record. If you try to put it in, the computer, when displaying data on the TV screen, thinks it has come to the start of the next file if it encounters this character. The result is it stops printing and only a portion of the file (up to the "*") will be displayed.

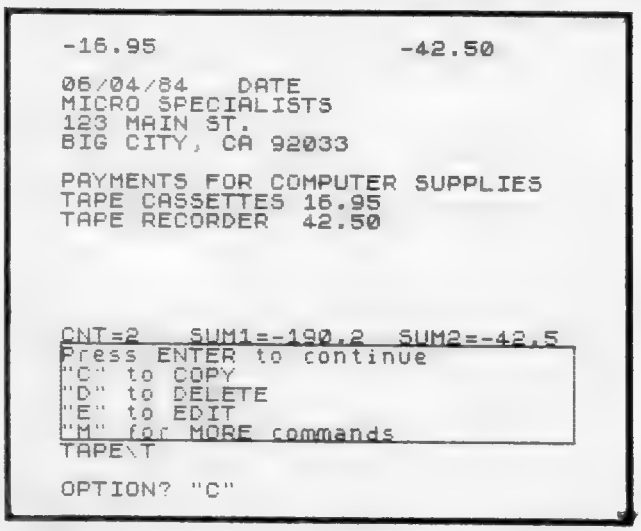
There is a tricky way around this dilemma, however. If you want to multiply, use the divide sign and write the expression a bit differently:

7*3=21 or 7/(1/3)=21 both mean the same thing

This enhancement looks for two numbers on the first line of a displayed file. These numbers can be added so they occupy either the left half of the screen or the right half, or both. When the computer finds a valid number in either or both of these locations, it evaluates the number and tallies it with all previous numbers found in the search. Three totals are printed with each display:

CNT= the number of files thus far found by a search
SUM1= the running total for all valid numbers on the left hand side of the top line of each file. This translates to numbers appearing in columns 1-16.
SUM2= the running total of numbers on the right side of line 1 (columns 17-32).

The exact positions for the numbers on line one are not critical as long as they're within the bounds noted above. In other words, the computer isn't fussy about where you stick your numbers (or even if you stick them at all), as long as the number that goes on the left does not extend into columns greater than 16 and the number that goes on the right starts somewhere to the right of column 17. A typical file display would look like this:



There are a few things you must watch out for. First, you must store numbers only in the positions for each number. If you put in something like: "101.23 credit card bill" on line one, the computer will ignore the amount. You are allowed digits from 0 to 9, and these symbols: (.) + / - You can have blank spaces both before and after the number you type in, but don't insert blanks between digits of a number. The digits "15" are interpreted as the number fifteen, but "1 5" is ignored and treated as zero.

Don't get the idea that you must have a number in every file. You can put other data in place of a number. The important thing to remember is if you want to utilize the math capabilities of one or both number tallies, you must have the number(s) in the correct position.

The second thing you must watch out for is typing numeric quantities in line one which are nonsensical. The computer stops dead in its tracks if you do. Examples of such erroneous entries include:

```
1//1     TWO "/"'s TOGETHER
..025   TWO "."'s TOGETHER
12/0    DIVIDE BY ZERO
1,000   COMMA USED IN NUMBER
```

Normally you would never enter numbers incorrectly, but typographic accidents will happen. If you do manage to enter a number that stops the computer, you will see an error code:

C Nonsense in BASIC, 9200:3
6 Number too big, 9200:3

To recover from this error, type LET Y$=" ": GO TO 1 and ENTER. This takes you back to the Main Menu where you should immediately look up the offending file and edit it.

Now I have told you what the modification does, where it looks for the numbers it is to process, and even what to watch out for in the way of data that won't work properly. What I haven't told you is how to implement this tally function. The way to do it is as follows:

After you have created a few files like the sample on the previous page, you can turn on the tally function by entering a search command followed by the reverse slash (on the "D" key) and the letter "T" for "tally".

Whenever the reverse slash-T combination is tacked onto the end of a search command, the search progresses as usual, but numbers, if they are in the right positions, are tallied. Totals are displayed with the regular display options. Using the "Micro Specialists" file as an illustration, you could search for:

TAPE\T	sum all records holding the name TAPE
MICRO SPECIALISTS\T	sum all payments to Micro Specialists
06/04\T	sum all transactions of June 4th
COMPUTER SUPPLIES\T	sum all payments for COMPUTER SUPPLI
06/ AND TAPE\T	sum all TAPE expenses in June (06/)
*\T	total all files

After each display, the current totals are given. Autosearches alphabetized in any desired way are perfectly legal, as are multi-word searches. If all you want are a display of the totals, and you do not particularly need to view individual files, you can do it by initiating an Autosearch. Type "Y" for PRINTOUT. Select the order and search parameters you need. Turn your printer OFF and run through the search. The computer will send files out to the printer just as always, but if it's off, nothing gets printed. The search progresses until it is complete, and you are left with the display options listing the total sums produced by the search.

Adding the TALLY enhancement requires modifying the Basic listing as well as adding a bit more machine code. We'll tackle the machine code first. Insert the short routine below that allows you to input the code:

```
9000 FOR X=63792 TO 63833
9005 PRINT X;" ";
9010 INPUT Y
9020 POKE X,Y
9030 PRINT Y
9040 NEXT X
9050 STOP
```

Then type GOTO 9000 to start it running. You will see a number printed on the screen and the computer waiting for you to input a number. One by one, input the numbers shown here which correspond to the address displayed on the TV screen. For example, when you see the number 63792 on the TV, you press 42 and ENTER.

63792	42	63813	190
63793	77	63814	48
63794	92	63815	9
63795	43	63816	62
63796	1	63817	57
63797	255	63818	190
63798	16	63819	56
63799	35	63820	242
63800	62	63821	35
63801	32	63822	16
63802	190	63823	243
63803	32	63824	201
63804	6	63825	62
63805	16	63826	32
63806	248	63827	190
63807	1	63828	32
63808	0	63829	233
63809	0	63830	35
63810	201	63831	16
63811	62	63832	250
63812	39	63833	201

After you input the last number (201 for address 63833) the computer will stop with a report **9-STOP** in line 9040. Then you must add or alter the Basic program listing so these lines are all incorporated into the Basic.

When you're finished, save a copy of both the updated machine code and the Basic by entering this command:

CLEAR: SAVE "p/f" CODE 63488,2046: SAVE "pro/file" LINE 9996

The tape you make will have the TALLY function added to it. The program will load and start like the master tape asking you if you wish to create a new data base or load an existing one. There are other ways to save this modification. I present this one here because it is the simplest. Read other sections on SAVING for other techniques. Reminder: Because of the length of this modification, you may wish to shorten the length of D$ as detailed at the start of this chapter. I recommend a length of 27020.

BASIC ALTERATIONS for
TALLY ENHANCEMENT

Incorporate these lines into Pro/File 2068's Basic:

```
   3 LET A=0: LET X$="": LET T=0
: LET T0=0
 103 LET T1=0: IF x$="AUTO" THEN
 GO TO 5200
 130 LET J$=CHR$ PEEK 23670
1020 IF A=0 AND B<P THEN LET T1=
T1+1
1049 IF CODE J$>32 AND E$(1, TO
18)<>D$(2 TO 19) THEN GO TO 8500
+CODE J$
1050 GO SUB 9100: PAPER 5: INK 0
: PRINT AT 16,0;"Press ENTER to
continue","""C"" to COPY",,"""D"
" to DELETE",,"""E"" to EDIT",,"
""M"" for MORE commands
  ": INPUT "OPTION? ";y$: IF y$=
"M" THEN PRINT AT 16,0;"""R"" to
 RE-INITIATE search","""N"" to s
tart NEW from main menu","""A""
to ADD new file","type a SEARCH
COMMAND, or": INPUT "OPTION?";y$
: IF y$="M" THEN GO TO 1051
1090 IF y$="R" THEN LET T=0: LET
 T1=0: LET T0=0: CLS : LET x$=x$
: GO TO 119+(A*5121)
7202 LET T1=T1+(B<P): IF Y=0 THE
N GO TO 1047
7229 IF J$="T" THEN GO TO 1049+(
Y$="C")
8583 GO TO 1
8584 GO TO 9200
8900 GO TO 1
9100 PRINT AT 15,0;"CNT=";T1;: I
F J$="T" THEN PRINT " SUM1=";T;"
  SUM2=";T0
9110 RETURN
9200 LET E$(1,1)=E$(1,1): IF USR
 63792 THEN LET T=T+VAL E$(1, TO
 16)
9205 LET E$(1,17)=E$(1,17): IF U
SR 63792 THEN LET T0=T0+VAL E$(1
,17 TO )
9220 GO TO 1050-(900*A*Y)
```

WHAT THE "TALLY" MACHINE
CODE DOES

The "TALLY" machine code is executed by Basic lines 9200 and 9205. The first statement of each of these lines sets the TS2068 system variable DEST equal to the address of the first, then the 17th characters of E$(1) respectively. This string is the file found by a search. The machine code starts at these addresses and checks each character looking for a number. As long as the program finds just the digits from 0-9 or the math symbols. parentheses, or the decimal point, the machine code returns to Basic with BC=00FF hex or 255 decimal. If some other character is found, BC will equal zero. Lines 9200 to 9205, therefore, test the validity of the possible numbers stored in the top line of a file. Their values are added to the totals if they're legal numbers. The computer skips over them if they're not.

MACHINE CODE DISASSEMBLY
OF "TALLY" ENHANCEMENT

Addr	Code	Label	Mnenomics	Comments
F930	2A4D5C	START	LD HL,(dest)	POINT TO ADDRESS OF E$
F933	2B		DEC HL	
F934	01FF10		LD BC,10FF	B COUNTS 10h SPACES, C<>ZER
F937	23	MORE	INC HL	
F938	3E20		LD A,20	CODE FOR "SPACE"
F93A	BE		CP (HL)	IS E$ CHAR. A SPACE?
F93B	2006		JR NZ,RPTE	NO? GO TO -RPTE-
F93D	10F8		DJNZ,MORE	YES? DO NEXT CHAR. UNLESS B
F93F	010000	EXIT	LD BC,0000	THEN MAKE C=0,
F942	C9		RET	AND RETURN TO BASIC
F943	3E27	RPTE	LD A,27	IS CHAR. A LEGAL SYMBOL?
F945	BE		CP (HL)	
F946	3009		JR NC,OUT_	NO? GO TO -OUT_-
F948	3E39		LD A,39	YES? IS IT A DIGIT 0-9?
F94A	BE		CP (HL)	
F94B	38F2		JR C,EXIT	NO? GO TO -EXIT-
F94D	23		INC HL	YES? CHECK NEXT CHAR.
F94E	10F3		DJNZ,RPTE	UNTIL B=0,
F950	C9		RET	THEN RETURN WITH C STILL <>
F951	3E20	OUT_	LD A,20	CHECK FOR TRAILING SPACES
F953	BE	OUT1	CP (HL)	ANYTHING BUT A SPACE RETURN
F954	20E9		JR NZ,EXIT	VIA -EXIT-
F956	23		INC HL	OTHERWISE RETURN WITH A
F957	10FA		DJNZ,OUT1	NON-ZERO VALUE IN
F959	C9		RET	C REGISTER

PRO/FILE 2068 COLOR CHANGES

If you're new to programming, playing with the colors is an excellent way to exert your influence over the computer. Pick a number and watch its effects. It may take several trys to get your colors coordinated just the way you want them, but mistakes will not alter Pro/File's operation other than to change a color. Even if you do get hopelessly stuck, you can always pull the plug on the computer and reload the original tape.

Color is used in Pro/File 2068 to provide interesting watching and to help you know which mode you are in or what menu you are looking at.

For example, the "status" numbers (OPEN, FILE, ORDER, and FORMAT) are printed with white characters in the Main Menu to set them off from the yellow and magenta colors of the other characters.

When you have the ADD/EDIT mode switched on, the edit cursor will be either yellow or white indicating whether newly typed characters will overwrite existing ones or be inserted between them.

The command menu in the ADD/EDIT mode is white letters on a blue background. Different screens of this menu toggle between bright and not bright to make the change from one to another easy to see.

When you make a search and display a file, it gets printed with green characters while the Display Options are black on a green background. If you edit a displayed file, unchanged parts retain their green color, but the altered characters turn yellow.

Different TV sets or monitors may have slight differences in color. Monochrome sets, of course, will show only different shades of the same color.

If you find my taste in colors just appalling you can change Pro/File's colors to fit your own needs/desires by going through the Basic listing and changing every PAPER and INK command to a different number of your own choosing. The following table lists every program line that sets color and explains their effects. To experiment with your own scheme, use a different number after the BORDER, INK, or PAPER command.

LINES IN PRO/FILE 2068 BASIC
WHICH ALTER COLOR

Line	Command	What it Does to the TV
5	BORDER 0 PAPER 0	Turns entire screen black
20	PAPER 3 INK 0 INK 6	Prints "Pro/File" title in black characters on a magenta background Prints prompts in yellow
30	INK 7 INK 5	Colors the rectangle white Resets subsequent printing to cyan
40	INK 3	Makes "STATUS" names magenta
50	INK 6	Prints Main Menu prompts yellow
102	INK 0 INK 7 INK 0	Draws rectangle black to erase it Resets subsequent printing to yellow Prints blank lines in black to erase
105	INK 5	Prints displayed files in cyan
1050	PAPER 5 INK 0	Puts Display Options on cyan background with black letters
1051	PAPER 0 INK 5	Displays "STATUS" on black background with cyan letters
1055	PAPER 0 INK 5	Restores black background and cyan letters for printing after the Display Options
1200	INK 0	Makes a black rectangle to erase old one
5004	INK 1 PAPER 6 PAPER 1 INK 7	Draws a blue line across top of EDIT Menu No effect. A remnant from earlier versions Makes ADD/EDIT menu blue background with white letters
5005	INK 6 PAPER 0	This is the flashing yellow edit cursor
5040	INK 6+I	Changes cursor from yellow to white (OVER to INSERT modes) NOTE:change to 2+I*5 for b&w TV's
5106	INK 7 PAPER 1	Prints the "<" or ">" in white on blue background
5200	PAPER 0 INK 7	Prompts (in AUTO mode) are printed in yellow on black
5210	INK 7 INK 6	Changes color of prompts to white...Then yellow in AUTO mode
5510	PAPER 5	Makes paper of "LOAD" prompt cyan
6500	INK 7	Prints the "DEFP" prompt in white
9850	INK 7 (four times)	"STATUS" values are printed in white

A PRO/FILE PRINT DRIVER MODIFICATION TO ENABLE FULL SIZE PRINTERS

This modification to Pro/File 2068 lets you connect the program up to full size printers. Instructions are given for the AERCO and TASMAN centronics interfaces, and the BYTE-BACK RS232 serial interface. What this modification does that the printer drivers which come with the interfaces won't do is allow you to "embed" special codes within the text to be lprinted.

Those not familiar with full size dot-matrix printers deserve a bit of an explanation here. Full size printers communicate with the computer through an interface. Generally, the system works such that you send a character code to the printer. The printer receives it and puts a figure representing that code on paper. Besides just printing characters, dot-matrix printers can perform all kinds of special functions. You can enlarge the size of printed characters, underline, double space, change from pica to elite, set tabs, print graphics, or any of a host of other niceties.

The difference between function characters and printed characters is the code number sent to the printer. The actual sending procedure is the same for both. Whenever the printer receives a control code, it interprets the code and turns on the function called for. The code (or command) does not actually get printed on the paper.

By convention, certain codes are reserved for functions. They cannot be used for anything else. In the case of the TS2068, these codes are listed as not used for printing on the TV. Thus, you can't just lprint a character to change some printer characteristic. You must use:

```
LPRINT CHR$ number
```

to send the code.

```
LPRINT "This printing is normal"
LPRINT CHR$ 14
LPRINT "This printing is Shifted Out"
```

is how you would send the SHIFT OUT control code. This technique works fine, but its limitation is that you must build in to your program every function change you wish to make. You cannot "embed" or place a special code directly within a string of text because the computer won't let you type the code in. In Pro/File 2068, everything sent to the printer goes in the form of a string so unless some additional software allows you to send control codes, fancy printing can not be executed.

This routine lets you "embed" control codes in text. You can, therefore, use special codes right in a file to make your printer do exactly what you want. In order to do this, you use the up pointing arrow (symbol shift-H) followed

by a 2 digit hex number representing the code you wish to send. For example:

```
LPRINT "This printing is normal
        ↑ØEThis printing is
        Shifted Out"
```

has the up arrow followed by "ØE", the 2 digit hex code that Shifts Out, or enlarges the characters to be printed. The command is "embedded" in the text. Any code can similarly be sent to the printer using the up arrow.

This modification was written so that it could be used with any interface with a minimum of additional software. Making the TS2068 output to printers other than the TS2040 printer requires 2 steps. First, you insert the actual machine code that activates the printer. Second, you change the channel table in ram so it stores the starting address of your printer routine.

The TS2068's ability to handle different inputs and outputs within its own operating system is a very nice feature of the computer and adds a tremendous amount of flexibility. Unfortunately, describing in detail how this system works is beyond the scope of this book.

Briefly, however, there exists in ram a table which the rom uses to decide where it should go to find instructions to execute all keyboard, TV screen, RAM extensions, and printer inputs and outputs. The start of this table is stored in the system variable, CHANS (address 23631 and 23632).

You can look at this table by running this short program. It determines the address of the channel table by peeking CHANS. Then it displays the 20 bytes of the table along with the PEEK value. The channel table furnishes the address of the input routine, the address of the output routine, and a channel specifier which tells what channel or device the input or output is going to.

```
10 LET T=0: LET X =PEEK 23631+256*PEEK 23632
20 FOR Z=X TO X+19: PRINT Z;"  ";PEEK Z;"    ";: LET T=T+1
30 IF T/5=INT (T/5) THEN PRINT CHR$ PEEK Z: NEXT Z
40 PRINT : NEXT Z
```

The table will look something like that shown on the next page. Five bytes are allotted for each channel. The first pair is the address of the channel output routine. The second pair is that of the input routine. The fifth byte is the specifier. This will hold the code for the letter -K- meaning "keyboard", -S- meaning "screen", -R- for "ram workspace", and -P- for "printer.

When you use a basic word like LPRINT, the rom checks the channel table and outputs whatever it is you want through the routine specified in the table. Thus, LPRINT calls for the -P- channel. The output routine is obtained from the first pair of bytes of the table listed for "P".

TS2068 RAM CHANNEL TABLE

Addr.	Val.		Specifier and/or meaning
26688	0		address of keyboard OUTPUT routine
26689	5		
26690	14		address of keyboard INPUT routine
26691	12		
26692	75	K	channel specifier -K-
26693	0		address of screen OUTPUT routine
26694	5		
26695	191		address of screen INPUT routine
26696	17		
26697	83	S	channel specifier -S-
26698	231		address of ram OUTPUT routine
26699	10		
26700	191		address of ram INPUT routine
26701	17		
26702	82	R	channel specifier -R-
26703	0		address of printer OUTPUT routine
26704	5		
26705	191		address of printer INPUT ROUTINE
26706	17		
26707	80	P	channel specifier -P-

To determine the actual address specified, take the value of the first byte and add to it the value of the second multiplied by 256. Therefore, you can find the printer output address by plugging in the values at 26703 and 26704:

0 + (5 * 256) = address 1280

Normally the addresses held in the channel table point to ROM routines, but you can change them by poking so they point to your own machine code. Then, every time a "stream" of data is to be sent out to a channel, your code will handle the transfer.

The listing that follows is made for use with the TASMAN interface. Alternate listings make it possible to use the AERCO and BYTE-BACK interfaces as well. Others will work if you change the output routine (labeled Pout) so it matches your port configuration.

To make this modification, load the Pro/File 2068 master tape as usual. When you have the Main Menu on the screen, exit the program by typing SHIFT 1, STOP, ENTER. Then add these program lines:

```
9000>FOR x=63672 TO 63788
9005 PRINT x;" ";
9010 INPUT y
9020 POKE x,y
9025 PRINT y
9030 NEXT x
9040 STOP
```

9000 to 9040 let you input the machine code given in the following listing. Once the lines are added, enter GOTO 9000 and type the value given below for each address. When you're finished, the program will stop and you can delete the lines if you wish.

MACHINE CODE POKER TABLE

to the address:	POKE the value:	to the address:	POKE the value:	to the address:	POKE the value:
63672	0	63711	79	63750	248
63673	0	63712	58	63751	201
63674	0	63713	186	63752	214
63675	254	63714	248	63753	55
63676	94	63715	254	63754	24
63677	40	63716	0	63755	211
63678	17	63717	40	63756	219
63679	79	63718	17	63757	191
63680	58	63719	58	63758	203
63681	185	63720	184	63759	71
63682	248	63721	248	63760	32
63683	254	63722	129	63761	181
63684	94	63723	79	63762	175
63685	40	63724	175	63763	211
63686	17	63725	50	63764	251
63687	205	63726	185	63765	61
63688	9	63727	248	63766	211
63689	32	63728	50	63767	123
63690	56	63729	186	63768	211
63691	64	63730	248	63769	251
63692	207	63731	50	63770	121
63693	12	63732	184	63771	211
63694	0	63733	248	63772	123
63695	0	63734	24	63773	62
63696	50	63735	207	63774	247
63697	185	63736	121	63775	211
63698	248	63737	7	63776	251
63699	175	63738	7	63777	62
63700	50	63739	7	63778	255
63701	186	63740	7	63779	211
63702	248	63741	50	63780	251
63703	201	63742	184	63781	121
63704	121	63743	248	63782	254
63705	254	63744	58	63783	13
63706	64	63745	186	63784	192
63707	48	63746	248	63785	14
63708	43	63747	60	63786	10
63709	214	63748	50	63787	24
63710	48	63749	186	63788	154

Those were the raw numbers. The next chart shows a more meaningful disassembly. In deciphering this code, a few things are important to remember. The computer comes here with the character to LPRINT stored in the accumulator. The routine executes once for each character to be printed out.

The print driver performs two functions. First it checks the character in the accumulator to see if it is an up pointing arrow. If it is, the next two characters sent are processed into a single code which has a value equal to the two hex digits processed. If the character is not an up-arrow, the second function is executed straight away: send the character to the printer.

Three bytes, addresses 63672, 63673, and 63674, are used by the driver to keep track of what's what during the interpreting of the control code sent following the up-arrow. In hex, these bytes equate to F8B8, F8B9, and F8BA. They are given the respective names, COMD (meaning command byte), FLBT (for flag byte), and CNT_(for counter byte).

The interpret function of this driver is the same whether you use the brand-X interface, brand-Y, or brand-Z. You'll notice that once interpreting is complete (or if it is not necessary), the routine jumps to the "printer out" function starting at F90C hex (63756 decimal). It is here where different interfaces require different programming. This is because each interface has its own ports and its own procedure for outputting to them. Therefore, you must choose the "Pout" routine you need based on which interface you use.

THE "UP-ARROW" PRINT DRIVER DISASSEMBLY

```
ADDR HEXCODE LABEL MNEMONIC
==========================
F8B8 00        COMD NOP           --Data bytes
F8B9 00        FLBT NOP           --
F8BA 00        CNT_ NOP           --
F8BB FE5E           CP 5E         -HERE TO START. IS CHAR. A "↑"?
F8BD 2811           JR Z,CTFL     -YES? GO TO -CTFL-  (F8D0h)
F8BF 4F             LD C,A        -NO. SAVE CHAR. IN C-REG.
F8C0 3AB9F8         LD A,(FLBT)   -PUT CONTENTS OF -FLBT- IN Acc.
F8C3 FE5E           CP 5E         -IS IT A "↑"?
F8C5 2811           JR Z,OFST     -YES?  GO TO -OFST- (F8D8h)
F8C7 CD0920    BRK? CALL 2009     -NO? IS BREAK KEY PRESSED?
F8CA 3840           JR C,Pout     -RETURNS-C- IF NOT. GOTO Pout.
F8CC CF             RST 08H       -IF NC, STOP w/REPORT-D
F8CD 0C              ERROR D                    "BREAK PRESSED"
F8CE 00             NOP
F8CF 00             NOP
F8D0 32B9F8    CTFL LD (FLBT),A   -HERE IF Acc="↑". PUT ↑ IN FLBT.
F8D3 AF             XOR A         -THEN ZERO THE Acc
F8D4 32BAF8         LD (CNT_),A   -AND MAKE CNT_ EQUAL ZERO
F8D7 C9             RET           -GET NEXT CHARACTER.
F8D8 79        OFST LD A,C        -IF FLBT="↑", GET CHAR IN Acc
F8D9 FE40           CP 40         -IS CODE>40h? GOTO F908h IF YES.
F8DB 302B           JR NC,F908

F8DD D630           SUB 30        -ELSE, TAKE AWAY 30. ANSWER IN C
F8DF 4F             LD C,A
F8E0 3ABAF8         LD A,(CNT_)   -CHECK COUNTER.
F8E3 FE00           CP 00         -IF ZERO, GOTO -COL1- (F8F8h)
F8E5 2811           JR Z,COL1
F8E7 3AB8F8         LD A,(COMD)   -PUT CONTENTS OF -COMD- IN Acc
F8EA 81             ADD A,C       -ADD IT TO C REGISTER
F8EB 4F             LD C,A
F8EC AF             XOR A         -ZERO ALL DATA BYTES
F8ED 32B9F8         LD (FLBT),A
F8F0 32BAF8         LD (CNT_),A
F8F3 32B8F8         LD (COMD),A
F8F6 18CF           JR BRK?       -AND GO TO -BRK?-  (F8C7h)
F8F8 79        COL1 LD A,C        -HERE IF COUNTER=ZERO.
F8F9 07             RLCA          -DIVIDE ANSWER IN C REGISTER
F8FA 07             RLCA           BY 16 (RESULT IS EITHER 1 OR 0)
F8FB 07             RLCA
F8FC 07             RLCA
F8FD 32B8F8         LD (COMD),A   -STORE RESULT IN -COMD-
F900 3ABAF8         LD A,(CNT_)   -INCREMENT COUNTER
F903 3C             INC A
F904 32BAF8         LD (CNT_),A
F907 C9             RET           -GET NEXT CHARACTER TO LPRINT
F908 D637           SUB 37
F90A 18D3           JR F8DF       -TAKE AWAY 37 AND GO BACK
```

NOTE: separate "printer-out" routines are given here for the TASMAN centronics interface, the AERCO centronics, and the BYTE-BACK RS-232 serial interface.

The routine you select depends on the interface you use. Regardless of the interface, however, the pokes to addresses from 63672 to 63755 (shown in the "poker table" are the same. The actual output routines given below all start at address 63756. This equates to F90C hex and is labeled "Pout" in the disassembly. The "poker table" lists the pokes necessary to use the Tasman interface. Alter the pokes to suit your needs if you use a different interface.

TASMAN Centronics
parallel interface

```
F90C DBBF      Pout IN A,(BF)   -PRINTER OUTPUT ROUTINE FOR
F90E CB47           BIT 0,A           >>>TASMAN INTERFACE<<<
F910 20B5           JR NZ,BRK?
F912 AF             XOR A       -PORT BFh IS CONTROL
F913 D3FB           OUT (FB),A
F915 3D             DEC A              INITIALIZE PIA CHIP
F916 D37B           OUT (7B),A
F918 D3FB           OUT (FB),A
F91A 79             LD A,C      -PUT CHARACTER IN Acc
F91B D37B           OUT (7B),A -AND SHIP IT OUT
F91D 3EF7           LD A,F7
F91F D3FB           OUT (FB),A
F921 3EFF           LD A,FF
F923 D3FB           OUT (FB),A
F925 79             LD A,C
F926 FE0D           CP 0D       -WAS CHARACTER SENT A CARRIAG
F928 C0             RET NZ       RETURN? GET NEXT CHAR. IF NO
F929 0E0A           LD C,0A     -ELSE, LOAD C REG. WITH CODE
F92B 189A           JR BRK?      LINE FEED AND SEND IT TOO.
```

AERCO Centronics
parallel interface

```
F90C DB7F      Pout IN A,(7F)   -PORT 7F hex IS CONTROL PORT
F90E CB67           BIT 4,A      ON INPUT; DATA PORT ON OUTPUT
F910 20B5           JR NZ,BRK? -IF PRINTER IS NOT READY TO
F912 CB4F           BIT 1,A      RECEIVE CHARACTER, GOTO -BRK'
F914 28B1           JR Z,BRK?
F916 79             LD A,C      -GET CHARACTER FROM C REGISTER
F917 D37F           OUT (7F),A -OUTPUT IT TO PRINTER
F919 FE0D           CP 0D       -IS IT "CARRIAGE RETURN"?
F91B C0             RET NZ      -GET NEXT CHARACTER IF NOT
F91C 0E0A           LD C,0A     -ELSE, SEND A LINE FEED TOO
F91E 18A7           JR BRK?
```

BYTE-BACK RS232
Serial Interface

```
F90C C5        Pout PUSH BC      -SAVE CHARACTER IN C-REGISTER
F90D 017F3F         LD BC,3F7F  -CONTROL PORT=3F7Fh
F910 ED78           IN A,(C)     -TEST PRINTER. IS IT READY TO
F912 CB7F           BIT 7,A       ACCEPT A CHARACTER?
F914 2003           JR NZ,F919  -YES? TEST MORE AT F919h
F916 C1             POP BC      -IF NOT, CLEAR THE STACK AND
F917 18AE           JR BRK?      GO BACK TO -BRK?-
F919 CB47           BIT 0,A     -TEST BIT-0
F91B 28F9           JR Z,F916   -IF IT'S ZERO, GO BACK
F91D 0D             DEC C       -POINT TO DATA PORT
F91E D1             POP DE      -GET CHARACTER FROM STACK
F91F 7B             LD A,E      -PUT IT IN ACCUMULATOR
F920 ED79           OUT (C),A   -AND SEND IT TO PRINTER
F922 FE0D           CP 0D       -WAS IT A CARRIAGE RETURN?
F924 C0             RET NZ      -RETURN FOR NEXT CHAR. IF NOT
F925 0E0A           LD C,0A     -IF YES, PUT A LINE FEED IN -C-
F927 189E           JR BRK?     -SEND IT OUT TOO.
```

"POKER TABLE" changes for
Byte-Back RS-232

address	value
63756	197
63757	1
63758	127
63759	63
63760	237
63761	120
63762	203
63763	127
63764	32
63765	3
63766	193
63767	24
63768	174
63769	203
63770	71
63771	40
63772	249
63773	13
63774	209
63775	123
63776	237
63777	121
63778	254
63779	13
63780	192
63781	14
63782	10
63783	24
63784	158

Changes for
Aerco Centronics

address	value
63756	219
63757	127
63758	203
63759	103
63760	32
63761	181
63762	203
63763	79
63764	40
63765	177
63766	121
63767	211
63768	127
63769	254
63770	13
63771	192
63772	14
63773	10
63774	24
63775	167
63776	0
63777	0

NOTE: If the above numbers do not work with your printer, try the entering them again, but enter the value of zero for addresses 63762 through 63765 inclusive.

To change the channel table so that the Timex operating system will send LPRINT data to this machine code, you need to add this Basic program line:

```
1 POKE 26703,187: POKE 26704,248
```

These numbers yield the address 187+(256*248) or 63675. Also note that if you use the Byte-Back serial interface you must add an additional line:

```
2 OUT 16255,0: OUT 16255,0: OUT 16255,0: OUT 16255,64:
  OUT 16255,206: OUT 16255,55
```

Once the channel table is poked with these numbers, the keyboard commands LPRINT and LLIST will obtain output instructions from the machine code which begins at 63675. The Timex 2040 printer will no longer work unless you change the table back to its original form. To do this you poke:

```
POKE 26703,0        and        POKE 26704,5
```

Finally, with the machine code printer driver added and Pro/File 2068's Basic changed to alter the channel table, you need to save the program on tape.

Put a fresh tape in the recorder and type the command:

```
CLEAR: SAVE "p/f"CODE 63488,2046: SAVE "pro/file"LINE 9996
```

Two programs will be saved. First the machine code with its additional print driver, then the Pro/File Basic with its new line 1. This will be your new master tape. To re-load it, power up your computer, set your PAPER and BORDER color to black, and enter this command:

```
CLEAR 63487: LOAD "p/f"CODE : LOAD "pro/file"
```

If you forget to turn the screen black before you try to load the program, the initial prompts will be invisible. Now when your interface is attached, printouts will go to your big printer. This printer routine does not require you to use any other software that comes with your interface. What's more, you could use this "driver" all by itself as an alternative to that provided by your interface supplier. Just save the machine code from address 63672 to 63788 on tape using the command:

```
SAVE "printer"CODE 63672,117
```

When you want to use it with another program, lower ramtop using CLEAR 63671. Then load the driver: LOAD "printer"CODE.
As long as you intend to use the driver with a program written in basic, or if machine code is used, the code doesn't overwrite the driver itself, you'll be able to use this routine anywhere you need to LPRINT. Remember though, LLIST won't work, nor will any procedure requiring tokens (words like PRINT, IF, AND, etc.) to be spelled out.

SUMMARY OF THE PROCEDURE
TO ADD BIG PRINTER FUNCTIONS

1. Read the chapters about how the program works, and how to modify the program.
2. Load your copy of Pro/File 2068
3. Break into the listing by typing SHIFT 1, STOP, ENTER.
4. Add lines 9000 to 9040 as listed in this chapter.
5. Type GO TO 9000 and start entering the numbers in the "Machine Code Poker Table", making the changes noted from addresses 63756 to 63788 for your interface.
6. If you use the Byte-Back or AERCO interfaces, your changes will not use as much memory as the TASMAN. When you enter the last number listed in the specified table, continue entering zeros until the last input for address 63788.
7. Add the BASIC line 1 to the program.
8. Save the program before you test it out. Enter:
   CLEAR: SAVE "p/f"CODE 63488,2046: SAVE "pro/file"LINE 9996
   This makes a new master tape.

```
   5 BORDER 0: PAPER 0: RANDOMIZ
E p: POKE 64035,PEEK 23670: POKE
 64036,PEEK 23671: LET a=0: LET
Y=0: DIM E$(15,32): CLS : RANDOM
IZE USR 64268
  20 PRINT AT 1,7; PAPER 3; INK
0;" PRO/FILE 2068 ": PRINT INK 6
;AT 3,4;"Separate MULTI-WORD";TA
B 4;"command words with the";TAB
 4;"token ""AND"""
  30 INK 7: PLOT 120,175: GO SUB
 9830: INK 5
  40 INK 3: PRINT AT 7,0;: GO SU
B 9850
  50 INK 6: PRINT AT 14,0;"
Type ""A"" to ADD files";TAB 4;"
""SAVE"" or ""LOAD"" for tape";T
AB 4;"""AUTO"" for AUTOSEARCH";T
AB 4;"""DEFP"" changes PRINT for
mat"
 100 INPUT "SEARCH COMMAND? ";X$
 102 IF x$="DEFP" THEN PRINT AT
14,0;: PLOT 120,70: GO SUB 6500:
 INK 0: GO SUB 9830: INK 7: PRIN
T AT 10,7;A$; INK 0;E$(1);E$(1);
E$(1): GO TO 50
 103 IF x$="AUTO" THEN GO TO 520
0
 105 INK 5: CLS : IF x$="A" THEN
 GO TO 5000
 106 IF x$="LOAD" THEN GO TO 550
0
 107 IF x$="SAVE" THEN SAVE f$ L
INE 1: GO TO 1
 110 IF x$="" THEN GO TO 10
 115 IF x$(LEN x$)=" AND " OR LE
N X$>32 THEN GO TO 10
 119 LET X$=X$
 120 RANDOMIZE USR 64040
 150 LET b=USR 64101
1000 PRINT AT 0,0;: DIM E$(15,32
): LET e$(1)="": IF USR 63575 TH
EN
1010 IF e$(1, TO 18)=d$(2 TO 19)
 AND b<>p THEN CLS : GO TO 150
1040 PRINT AT 21,0;x$
1045 IF A THEN GO TO 7000
```

-Clean house. Set screen colors, Put P's value in 64035-6, Turn OFF autosearch (A=0), Turn OFF printer output (Y=0), Reset file display store [E$(15,32)], reset pointers in data (USR 64268).

-The MAIN MENU

-Get a rectangle

-Get current "status"

-Here to enter Search Command or special functio

-Function DEFP gosubs to 6500 to get printer format.
-E$(1) three times=3 blank lines. Used to erase.

-Function AUTO initialized at 5200

-Function "A" (for ADD) starts at 5000

-Function LOAD is at 5500.

-Function SAVE is done right here.

-Invalid entries go back to reinput.

-Start search here
-Put X$ in search buffer
-Look for it
-Print it and put it in E$

-If "SEARCH IS COMPLETE" comes up before its tim go back to 150 to search more.

-AUTOSEARCHES GO TO 7000.

```
1050 PAPER 5: INK 0: PRINT AT 16
,0;"Press ENTER to continue","""
C"" to COPY",,"""D"" to DELETE",
,"""E"" to EDIT",,"""M"" for MOR
E commands            ": INPUT "O
PTION? ";y$: IF y$="M" THEN PRIN
T AT 16,0;"""R"" to RE-INITIATE
search","""N"" to start NEW from
 main menu","""A"" to ADD new fi
le","type a SEARCH COMMAND, or":
 INPUT "OPTION?";y$: IF y$="M" T
HEN GO TO 1051
1051 IF Y$="M" THEN PAPER 0: INK
 5: PRINT AT 16,0;: GO SUB 9850:
 PRINT """M"" for MORE commands"
,: INPUT "OPTION? ";Y$: IF Y$="M
" THEN GO TO 1050
1055 PAPER 0: INK 5
1060 IF y$="" AND b<p THEN CLS :
 GO TO 150
1070 IF y$="N" THEN GO TO 1
1075 IF USR 64268 THEN
1080 IF PEEK 64026+256*PEEK 6402
7<>PEEK 23627+256*PEEK 23628+6 A
ND (y$="D" OR y$="E") THEN GO TO
 4000
1090 IF y$="R" THEN CLS : LET x$
'=x$: GO TO 119+(A*5121)
1100 IF y$="C" THEN GO TO 7205
1200 IF y$="DEFP" THEN PRINT AT
20,0,,,,: PRINT AT 16,0;: PLOT 1
20,50: GO SUB 6500: INK 0: GO SU
B 9830: LET y$="M": GO TO 1051
1500 IF y$<>"" AND y$<>"E" AND y
$<>"D" THEN LET x$=y$: GO TO 105
2000 GO TO 1050
4000 LET p=p-USR 63640: IF y$="D
" THEN GO TO 1
4010 GO TO 5002
5000 DIM e$(15,32): CLS
5002 LET m=0: LET z=23658: LET i
=0: LET l=0: LET c=0
5004 INK 1: PLOT 0,48: PAPER 6:
DRAW 255,0: INK 0: PRINT AT 16,0
; PAPER 1; INK 7;"CURSOR CONTROL
S: ARROWS move                curs
or one position   ENTER: Next li
ne                SHIFT 1:Altern
ate Insert/Over   SHIFT 0:Delete
 character        SHIFT 9:More c
ommands           ";AT l,c;: LET
 m=0
```

-Otherwise print Display Options

-Selection goes into Y$

-Just ENTER goes back to search more.

-Select "N" to go back to Main Menu.
-Reset pointers in data.
-Select "D" or "E" to go to EDIT routine at 4000.

-Select "R" to restart same search. When A=0, go to 119. If A=1, go to 5240.
-Select "C" to lprint at 7205.
-DEFP changes print format as before. Then you go back to Display Options.

-Words other than "", "E", or "D" are treated as a new Search command.

-Delete file on screen here.

-When EDITING, keep the file in E$.
-Else, clear the array for ADD.
-Initialize ADD/EDIT variables.

-Print EDIT commands

```
5005 INK 6: PAPER 0: FLASH 1: PR
INT AT l,c;SCREEN$ (l,c)
5007 ON ERR GO TO 5005
5010 IF INKEY$="" THEN GO TO 501
0
5016 IF INKEY$=" STOP " THEN ON
ERR RESET : FLASH 0: GO TO 6000
5020 LET y$=INKEY$: FLASH 0: PRI
NT AT l,c;SCREEN$ (l,c): IF CODE
 y$<16 THEN GO TO 5100+CODE y$
5025 IF i THEN LET e$(l+1)=e$(l+
1, TO c)+" "+e$(l+1,c+1 TO 31):
PRINT AT l,0;e$(l+1)
5030 PRINT AT l,c;y$: LET e$(l+1
,c+1)=y$: LET c=(c+1)*(c<31): LE
T l=l+(c=0): LET l=l*(l<15): FLA
SH 1: PRINT AT l,c;SCREEN$ (l,c)
5040 PRINT INK 6+i;AT l,c; FLASH
 1;SCREEN$ (l,c): FOR x=1 TO 5:
NEXT x
5103 GO TO 5010
5104 FOR x=1 TO 13: LET e$(x+1)=
e$(x+2): PRINT AT x,0;e$(x+1): N
EXT x: LET e$(15)="": PRINT e$(1
5): GO TO 5040
5105 LET e$(l+1,c+1 TO )="": PRI
NT AT l,0;e$(l+1): GO TO 5040
5106 POKE z,(PEEK z=0)*8: PRINT
AT l,c; INK 7; PAPER 1;CHR$ (60+
(PEEK z/4)): PAUSE 20: PRINT AT
l,c;e$(l+1,c+1): GO TO 5040
5107 LET i=NOT i: GO TO 5040
5108 LET c=c-(c>0): GO TO 5040
5109 LET c=c+(c<31): GO TO 5040
5110 LET l=l+(l<14): GO TO 5040
5111 LET l=l-(l>0): GO TO 5040
5112 LET e$(l+1)=e$(l+1, TO c)+e
$(l+1,c+2 TO 31)+" ": PRINT AT l
,0;e$(l+1): GO TO 5108
5113 LET c=0: LET l=l+1: LET l=l
-(l>14): GO TO 5040
5114 FOR x=15 TO l+2 STEP -1: LE
T e$(x)=e$(x-1): PRINT AT x-1,0;
e$(x): NEXT x: LET e$(l+1)="": P
RINT AT l,0;e$(l+1): GO TO 5040
5115 IF m=0 THEN  PRINT BRIGHT 1
; PAPER 1;AT 16,0;"Press STOP to
 close file          SHIFT-2 for C
APS lock             SHIFT-3 for L
INE DELETE           SHIFT-4 for L
INE ERASE            SHIFT-SYMBOL
SHIFT: line insert ": LET m=-1:
GO TO 5040
```

-Start cursor blinking.

-Keep BREAK key from forcing a halt.
-Scan for key press.

-Exit ADD/EDIT mode when STOP is pressed.

-If command key is pressed, execute at the proper line (5104-5115).

-When INSERT mode is ON, move everything on line over one space.

-Put key being pressed on screen and in E$. Then move the cursor to next spot.

-Flash it and pause briefly.

-Then go back for next character.
-DELETE LINES here.

-ERASE LINES here.

-Shift from upper/lower case here.

-INSERT/OVER modes toggle here.
-Cursor LEFT.
-Cursor RIGHT.
-Cursor DOWN.
-Cursor UP.
-DELETE character.

-Carriage Return (NEXT LINE)

-INSERT LINE here.

-MORE commands.

```
5116 IF m=-1 THEN PRINT AT 16,0,
,: GO SUB 9850: LET m=1: GO TO 5
040                                  -"STATUS" Display
5117 GO TO 5004
5200 LET a=1: PRINT AT 14,0; PAP     -AUTOSEARCH is initialized here. A=1
ER 0; INK 7;" PRINT OUT? (Y/N)",      when AUTO mode is on.
,,,,,,,: INPUT Y$: LET Y=Y$="Y"      -Y=1 when lprinting is ordered.
5210 PRINT AT 16,0; INK 7;" SEAR
CH COMMAND?",,, INK 6;"Press jus
t ENTER to return to   Main Menu
": INPUT X$: IF X$="" OR LEN X$>     -Search Command goes into X$ as usual.
32 THEN GO TO 1
5220 LET Z$="0": PRINT AT 14,0;"     -Specify file line by which ordering
TYPE LINE NUMBER BY WHICH FILES       occurs here.
WILL BE ORDERED. Type ""0"" if
  ordering is not required": INP
UT Z$: IF z$="" THEN GO TO 1
5222 ON ERR GO TO 5220: IF VAL Z     -Bad entries go back to 5220 to redo.
$)=0 AND VAL Z$<=15 THEN GO TO 5
230
5225 GO TO 5220
5230 ON ERR RESET : LET S=VAL Z$     -Valid entries are placed in S.
: CLS : IF S=0 THEN LET X$=X$: G      S=0 when ordering is not required so
O TO 119                             proceed with search at 119.
5240 LET P$="      ": LET Q$="zzz    -"ordering strings", P$ and Q$ point to
zz": LET X$=X$: GO TO 119             lowest and highest in order. Then search.
5500 PLOT 120,50: GO SUB 9830: P     -Function LOAD starts here.
RINT AT 16,1;"HAS THIS FILE BEEN
 SAVED?(Y/N)": INPUT y$: IF y$<>
"Y" THEN GO TO 1
5510 PRINT AT 16,1; PAPER 5; INK     -Loading new data loads the Basic and
 0;"WHAT FILE NAME DO YOU WISH        the files. It DOES NOT load the machine
  ";AT 17,1;"TO LOAD ";: INPUT F     code.
$: PRINT f$: LOAD F$: GO TO 1
6000 FOR x=1 TO 15                   -Files are placed in D$ data array when
6004 LET e$(x)=e$(x)                  they are closed from ADD/EDIT mode.
6005 IF USR 63489 THEN               -Transfer from E$ to D$ here.
6010 NEXT x
6020 LET p=USR 63530                 -Update variable P.
6030 GO TO 1                         -Then back to Main Menu.
6500 PRINT INK 7;" ENTER DESIRED     -DEFP function is carried out here.
 FORMAT. Type ALL ";" or line nu
mbers separated by"," ""/"". Use
 ""0"" to print blank"," lines
                        ": GO SU
B 9830
6510 LET C$="": LET n=0: INPUT "
format?";a$: LET a$=a$+"/": IF a     -Format goes into A$. Then check syntax
$(1)="/" THEN GO TO 6510              of input.
6515 IF A$="ALL/" THEN FOR X=1 T     -If you type "ALL", create C$ now.
O 15: LET C$=C$+CHR$ X: NEXT X:
LET D=3: GO TO 6600
```

```
6520 LET D=1: FOR x=1 TO LEN a$:
 IF a$(x)<"/" OR a$(x)>"9" THEN
GO TO 6510
6525 IF X<LEN A$ THEN IF A$(X TO
 X+1)="//" THEN GO TO 6510
6530 IF a$(x)="/" THEN LET n=VAL
 a$(d TO x-1): LET d=x+1: IF n<1
6 THEN LET c$=c$+CHR$ n
6535 IF n>15 THEN GO TO 6510
6540 NEXT x
6600 RETURN
7000 IF S=0 THEN GO TO 7202
7005 IF B=p THEN GO TO 7070
7010 LET S$=E$(S, TO 5)
7020 IF S$=P$ THEN GO TO 7200
7030 IF S$>P$ THEN IF S$<Q$ THEN
  LET Q$=S$
7050 GO TO 7230
7070 IF P$="zzzzz" THEN GO TO 10
50
7080 LET P$=Q$: LET Q$="zzzzz":
CLS : LET X$=X$: GO TO 119
7200 POKE PEEK 64026+256*PEEK 64
027,0
7202 IF Y=0 THEN GO TO 1050
7205 FOR X=1 TO LEN C$
7210 IF CODE C$(X)=0 THEN LPRINT

7220 IF CODE C$(X) THEN LPRINT E
$(CODE C$(X))
7225 NEXT X
7230 IF A AND B<p AND Y$<>"C" TH
EN CLS : GO TO 150
7240 GO TO 1050
9830 DRAW 134,0: DRAW 0,-48: DRA
W -254,0: DRAW 0,48: DRAW 120,0:
 RETURN
9850 PRINT "OPEN:  "; INK 7;LEN
9850 PRINT "OPEN:  "; INK 7;LEN
d$-p;: PRINT " bytes";TAB 0;: PR
INT "FILE:  "; INK 7;f$;TAB 0;:
PRINT "ORDER: "; INK 7;s;TAB 0;:
 PRINT "FORMAT:"; INK 7;a$;TAB 0
;: RETURN
9996 DIM d$(28020): LET p=20: LE
T d$( TO 20)="*SEARCH IS COMPLET
E*": LET a$="ALL": LET c$="": FO
R x=1 TO 15: LET c$=c$+CHR$ x: N
EXT x: LET s=0
```

-Check syntax some more and redo if nec
essary.

-Else put the NUMBER typed in A$ into
the CHARACTER of C$.
-C$ tells the printer what to do, A$
tells you what its going to do.

-AUTOSEARCHES come here.

-Put the line to order by in S$.
-If its in correct order go to 7200.
-Otherwise advance "ordering strings".

-Go to Display Options if ordering is
complete.
-Else, jockey "ordering strings" and
return to search more.
-"Tag" the file that's ready to be
displayed.
-Go to Display Options for TV displays
-Here for output to printer. Format is
in C$. If character in C$ has code of
0, then lprint a blank space.
-Otherwise print the line specified by
the code of the character in C$.
-Lprint next line.
-If AUTOSEARCH on and search is not
complete then continue searching.
-Else, go print Display Options.
-RECTANGLE drawing subroutine.

-"STATUS" subroutine.

-Pro/File INITIALIZATION. When you
load the master tape, it starts run-
ning at 9996. D$(28020) stores all
files. All other "STATUS" variables
are also created.

```
9997 PRINT AT 19,2; INK 7;"Press
""C"" to CREATE a new file   or
""L"" to LOAD an existing one":
INPUT y$: PRINT AT 19,0;D$(100 T
O 164): IF y$="L" THEN GO TO 551
0
9998 PRINT AT 19,4; INK 7;"ENTER
 A NAME FOR THIS FILE": INPUT F$
: IF f$="" OR LEN f$>10 THEN GO
TO 9998
9999 GO TO 1
```

-CREATE NEW or LOAD EXISTING FILE option is executed here.

-File name is input here.

-Go to Main Menu.

PRO/FILE 2068's BASIC VARIABLES
What they are and How they're used

- D$ -a large string array consisting of 28020 spaces. This is where all your files are stored.
- P -The number which "points" to the last character of D$ used to store data. As you add more files, the value of P increases.
- A$ -A string which displays the current print format.
- C$ -an equivalent to A$ which the computer uses to determine which file lines shall be printed.
- X -a control variable in FOR.... NEXT.... loops
- S -represents the line number by which files are ordered in an AUTOSEARCH.
- F$ -the name given to a particular program and its data.
- X$ -used to store a search command or special function.
- A -a flag set to 1 when Autosearch is ON. A=0 when its OFF.
- E$ -an array consisting of 15 lines, each holding 32 spaces. It corresponds to a screen display of a file. Used to store individual files when adding or editing them.
- B -created by the machine code search routine. Counts the number of bytes left to search.
- Y$ -the selection from the Display Options.
- M -used in ADD/EDIT mode to determine which "menu" is to be displayed.
- Z -stores the address 23658, the TS2068 system variable called FLAGS2.
- L -equals the LINE on which the EDIT cursor blinks.
- C -the COLUMN on which the EDIT cursor blinks.
- I -INSERT mode switch. I=1 when Insert mode is ON.
- Z$ -used in selecting a line number for ordering in an AUTOSEARCH.
- N -also used in selecting a line number for ordering.
- P$ -an "ordering string" used to order files in an AUTOSEARCH.
- Q$ -like P$. Used to find the next file in alphabetical order.
- D -a pointer used in calculating printer format.

You will notice when you look at the Basic listing that the variables above can be referred to in either upper or lower case. The computer does not distinguish between the two.

This behavior is altogether different when the machine operates on the contents of a string. Here, lower case cannot be substituted for upper case characters.

APPENDIX II PRO/FILE 2068 MACHINE CODE DISASSEMBLY

Machine language in Pro/File 2068 is located above Ramtop. It starts at address F801 hex (63489 decimal) and extends to FB1B hex (64283 dec.), a total of 794 bytes. Not all of this area is used, however. Actual code runs from:

F801 to F845 hex (63489-63557 dec.)
F857 to F880 hex (63575-63676 dec.)
F898 to F8B7 hex (63640-63671 dec.)
FA28 to FB1A hex (64040-64283 dec.)

An area from F9E2 to FA27 (63970-64039) is used by the program to store search commands and various data.

This leaves two small blocks of memory, F846 to F856 (63558-63574) and F8B8 to F9E1 (63672-63969) free. Also, all memory above FB1A (64283) is not used by the program and is available for additional programming.

The charts that follow detail Pro/File's action. These disassemblies appear in a slightly different format than those shown in the previous chapter. Four columns are used.

Column 1

This column shows the hex address of the machine code instruction.

Column 2

This gives the hex machine code. The number here can be from 1 to 4 pairs of hex digits. The entire instruction is therefore written on one line. In the last chapter, a command like LD BC, 0001 took up three lines of a chart. Here, each pair of digits for the entire command is printed on the same line.

Column 3

This is the "LABEL" column. If an address of machine code is referred to in several places throughout the program, the address is given a name which is listed in this column. Thinking of an address by its name rather than by its number facilitates understanding the action of the program without needing to continually stop to look up the significance of an address.

Column 4

Shows the mnenomic representation of the code. Whenever a labeled address is used in this column, the label is printed rather than the actual hex number.

PRO/FILE WORKSPACE
ADDRESSES AND THEIR USES

Address Hex	Decimal	Label Name	What it's used for
F9E2	63970	SBUF	The "Search Buffer". This address is the starting address of 50 bytes used to hold the current search command.
FA16	64022	DPTR	Pronounced "D-Pointer". The first of a two byte variable used to store the address of the last character in the D$ array to be checked in a search.
FA18	64024	DCNT	Called "D-Count". Stores the number of characters checked in a search.
FA1A	64026	FADR	"File Address". These bytes store the starting address of a file found in search.
FA1C	64028	FEND	"File End". Stores the address of the end of a file found in a search.
FA1E to FA20		.....	Not Used
FA21	64033	SWRD	"Search Word" points to the address of the current search word in multi-word search commands.
FA23	64035	PPTR	"P-Pointer" stores the total number of characters used to store data.
FA25	64037	SADR	"Search Address" stores the current byte in SBUF being matched in a search.
FA27	64039	.....	A single byte. Not Used.

With the exception of SBUF and those addresses marked "Not Used", all of the above mark the first byte of a two byte variable used to store an address or other large number. They are used in a way similar to the "System Variables" used by the computer's ROM operating system. (See Appendix D, page 262, of the Timex User Manual)

Disassemblies are described in the order in which they are called by the Basic program. This is not necessarily the order in which they are located in memory.

CLRZ (USR 64268)

The "Clear Z" routine is the first machine code called. In line 5 of the Basic, the statement RANDOMIZE USR 64268 jumps the computer to code located at FB0C hex.

This routine scans the D$ array and every time a CHR$ 0 is found, it changes it into a "*" or a CHR$ 2A hex. This code serves to clean up D$ before going into a search. A previously run ordered search can leave D$ with zeros in place of the asterisks which mark the start of each individual file. The search routine expects asterisks.

```
FB0C CD5BFA   CLRZ CALL GET1          MAKES HL=ADDRESS OF D$(1.
FB0F ED4B23FA      LD BC,(PPTR)       BC=NO. OF BYTES USED FOR DATA
FB13 AF            XOR A              MAKES ACCUMULATOR=ZERO
FB14 EDB1          CPIR               SCAN DATA FOR ZEROS
FB16 E0            RET PO             RETURN WHEN FINISHED
FB17 2B            DEC HL             ELSE, POINT TO THE ZERO IN D$
FB18 362A          LD (HL),2A         LOAD THAT ADDRESS WITH 2A hex
FB1A 18F8          JR FB14            GO BACK TO HUNT SOME MORE
```

PUTB (USR 64040)

Basic line 120, RANDOMIZE USR 64040, puts the search command (X$) into the Search Buffer. It presets the Pro/File system variables, DCNT, SADR, and DPTR with their initial values needed to start a search. Finally, the routine checks the second to the last character of X$ to see if it is a 5C (the reverse slash--on the "D" key). If it is, the routine returns to Basic with BC=the code for the last character of X$. If it isn't, BC=1 when the return to Basic is made.

```
FA5B ED4B23FA   PUTB LD HL,(PPTR)     NO. OF BYTES USED IN D$
FA5F 2218FA          LD (DCNT),HL     PUT IT IN -DCNT-
FA62 21E2F9          LD HL,SBUF       GET START OF BUFFER
FA65 2225FA          LD (SADR),HL     PUT IT IN -SADR-
FA68 2A4F5C          LD HL,(dest)     ADDR. OF SEARCH COMMAND (X$)
FA6B 010400          LD BC,0004       SKIP OVER POINTERS
FA6E 09              ADD HL,BC

FA6F 11E2F9          LD DE,SBUF       DE=ADDRESS OF DESTINATION
FA72 ED4B7D5C        LD BC,(stln)     BC=NO. OF BYTES TO MOVE
FA76 EDB0            LDIR             TRANSFER X$ INTO SEARCH BUFFER
FA78 3E5C            LD A,5C          THE CODE FOR "\"
FA7A 12              LD (DE),A        PUT IT AT END OF SEARCH COMMAND
FA7B C5              PUSH BC          SAVE BC AND HL
FA7C E5              PUSH HL

FA7D CD5BFA          CALL GET1        GET ADDR. OF FIRST CHAR. OF D$
FA80 2216FA          LD (DPTR),HL     PUT IT IN -DPTR-
FA83 E1              POP HL           RESTORE OLD VALUES OF BC AND HL
FA84 C1              POP BC
FA85 03              INC BC           MAKE BC=1
FA86 2B              DEC HL           BACK UP 2 SPACES
FA87 2B              DEC HL
FA88 BE              CP (HL)          COMPARE Acc. WITH THAT IN SBUF
FA89 C0              RET NZ           IF IT'S NOT A "\" THEN RETURN
FA8A 23              INC HL           ELSE, GET NEXT CHARACTER
FA8B 4E              LD C,(HL)        PUT IT IN C-REGISTER
FA8C C9              RET              AND GO BACK TO BASIC
```

GET1 (Address 64091 decimal)

This subroutine is not called from Basic, but is, rather, used several times by other machine code routines. GET1 is very short and simple. It finds the address of the first character of the D$ array. When the return is made, the address is located in the HL register.

```
FA5B 2A485C    GET1 LD HL,(vars)    ADDRESS OF VARIABLES AREA
FA5E 010600         LD BC,0006
FA61 09             ADD HL,BC       ADD 6 TO SKIP OVER POINTERS
FA62 C9             RET             RETURN WITH HL HOLDING THE
                                       ADDRESS OF THE FIRST CHAR.
                                       OF D$ ARRAY
```

SRCH (USR 64101)

This is the search routine, the powerhouse of the program. When Basic line 150 sends the computer to FA65 hex, this routine takes the characters held in the search buffer and checks them against every character in the D$ array.

When the computer comes across the AND token in a search command, the individual file being scanned has its beginning and end points loaded into the Pro/File system variables FADR and FEND respectively. Then the individual file is checked for matches to subsequent words in the search command.

The sequence continues as long as matches are found, but if the computer finds a non-match, it goes back to searching more of D$ for matches to the first word of the search command.

Two events will cause this routine to return to Basic. If the reverse slash in the search buffer is encountered, it means a match to the entire search command was found and the program returns to Basic. If all the data in D$ is checked, but the slash was never reached, it means that no match was found.

In either case, the routine goes back to Basic, but before it does, the Pro/File system variable FADR is loaded with the address of the start of a file. In the case of a match, the address of the file containing the match is loaded. Non-matches result in FADR holding the address of the first file in D$. This file says, "SEARCH IS COMPLETE".

The other system variables are also updated before returning. Besides FADR, FEND points to the end mark of the file, DCNT stores the number of bytes in D$ still unchecked, DPTR has the address of the last D$ character checked, and SWRD keeps the address of the current word in the search buffer.

Therefore, these variables keep track of how far along a search has progressed. When the search routine returns to Basic, it has a file and it knows where it stopped. In this way, you can very easily go back into the search and simply pick up where you left off.

Also note that you can return with BC equal to PPTR if a non-match causes a return to Basic, or BC equal to DCNT if a match is found. Since line 150 says LET B=USR 64141, the variable B will equal this number.

SRCH (USR 64101)

```
FA65 2A18FA    SRCH  LD HL,(DPTR)
FA68 ED4B18FA        LD BC,(DCNT)
FA6C ED5B25FA  STRT  LD DE,(SADR)
FA70 1A              LD A,(DE)
FA71 EDB1            CPIR
FA73 E2A0FA          JP PO,OUT_
FA76 2216FA          LD (DPTR),HL
FA79 ED4318FA        LD (DCNT),BC
FA7D 13        NEXT  INC DE
FA7E 1A        CHEK  LD A,(DE)
FA7F FEC6            CP C6
FA81 280C            JR Z,AND_
FA83 FE5C            CP 5C
FA85 281A            JR Z,LAST
FA87 EDA1            CPI
FA89 E2A0FA          JP PO,OUT_
FA8C 28EF            JR Z,NEXT
FA8E 18D5            JR SRCH
FA90 CDE1FA    AND_  CALL FILE
FA93 CDC1FA          CALL AND2
FA96 2A18FA          LD HL,(DPTR)
FA99 ED4B18FA        LD BC,(DCNT)
FA9D 28DF            JR Z,CHEK
FA9F 18C4            JR SRCH
FAA1 CDE1FA    LAST  CALL FILE
FAA4 2A1AFA          LD HL,(FADR)
FAA7 AF              XOR A
FAA8 BE              CP (HL)
FAA9 28BA            JR Z,SRCH
FAAB C9              RET
FAAC CD5BFA    OUT_  CALL GET1
FAAF 23              INC HL
FAB0 CDE1FA          CALL FILE
FAB3 011200          LD BC,0012
FAB6 ED42            SBC HL,BC
FAB8 ED4B23FA        LD BC,(PPTR)
FABC ED4318FA        LD (DCNT),BC
FAC0 C9              RET
FAC1 2A1CFA    AND2  LD HL,(FEND)
FAC4 ED4B1AFA        LD BC,(FADR)
FAC8 C5              PUSH BC
FAC9 ED42            SBC HL,BC
FACB E5              PUSH HL
FACC C1              POP BC
FACD E1              POP HL
FACE ED5B21FA  LOOK  LD DE,(SWRD)
FAD2 13        WD2_  INC DE
FAD3 1A              LD A,(DE)
FAD4 FEC6            CP C6
FAD6 C8              RET Z
FAD7 FE5C            CP 5C
FAD9 C8              RET Z
FADA EDA1            CPI
FADC E0              RET PO
FADD 28F3            JR Z,WD2_
FADF 18ED            JR LOOK
FAE1 E5        FILE  PUSH HL
FAE2 2B        BACK  DEC HL
FAE3 7E              LD A,(HL)
FAE4 FE2A            CP 2A
FAE6 2804            JR Z,CONT
FAE8 FE00            CP 00
FAEA 20F6            JR NZ,BACK
FAEC 221AFA    CONT  LD (FADR),HL
FAEF E1              POP HL
FAF0 7E        UP1_  LD A,(HL)
FAF1 FE2A            CP 2A
FAF3 2808            JR Z,FND_
FAF5 FE00            CP 00
FAF7 2804            JR Z,FND_
FAF9 23              INC HL
FAFA 0B              DEC BC
FAFB 18F3            JR UP1_
FAFD 221CFA    FND_  LD (FEND),HL
FB00 ED4318FA        LD (DCNT),BC
FB04 2216FA          LD (DPTR),HL
FB07 ED5321FA        LD (SWRD),DE
FB0B C9              RET
```

```
HL=START ADDR. OF D$ CHARACTER
BC=NO. OF BYTES TO CHECK
DE=ADDR. OF CHARACTER IN -SBUF.
PUT CHARACTER IN ACCUMULATOR
SCAN D$ FOR A MATCH
IF FINISHED, GOTO "OUT_"
SAVE LAST D$ ADDRESS CHECKED
SAVE BYTES STILL TO GO
GET NEXT CHARACTER FROM "SBUF"
PUT IT IN ACCUMULATOR
IS IT "AND"?
IF YES  GOTO "AND_"
IS IT A SLASH?
IF YES  GOTO "LAST"
ELSE, COMPARE W/NEXT D$ CHAR.
IF FINISHED, GOTO "OUT_"
MATCHES GO BACK TO "NEXT"
NO MATCH? START OVER
SAVE FILE START, END, etc.
IS NEXT WORD ALSO IN FILE?
RESTORE HL AND BC TO PROPER
        VALUES
MATCHES GO TO "CHEK"
OTHERWISE, START OVER
SAVE FILE START, END, etc.
PUT ADDRESS OF START IN HL
ZERO THE ACCUMULATOR
IS FIRST FILE CHAR. A ZERO?
IF YES, CONTINUE SEARCHING
IF NOT, BACK TO BASIC
GET ADDRESS OF D$(1)
POINT TO SECOND CHARACTER
GET FILE START, END, etc.


BC=NUMBER OF BYTES USED IN D$
PUT SAME VALUE IN "DCNT"
GO BACK TO BASIC
HL=ADDRESS OF END OF FILE
BC=ADDRESS OF START
SAVE START
DETERMINE LENGTH
STACK IT
POP IT INTO BC
PUT STARTING ADDRESS IN HL
DE=ADDRESS OF "AND" TOKEN
POINT TO START OF NEXT WORD
PUT THE CAHRACTER IN ACCUMULATOR
IS IT "AND"?
IF YES, RETURN
IS IT A SLASH?
IF YES, RETURN
ELSE, COMPARE
IF FINISHED, RETURN
IF CHARACTERS MATCH, DO IT AGAIN
ELSE, KEEP LOOKING FOR FIRST
SAVE ADDRESS OF LAST COMPARISON
STEP BACK
PUT CHARACTER IN ACCUMULATOR
IS IT AN ASTERISK?
YES, GOTO "CONT"
IS IT A ZERO?
IF NOT STEP BACK FURTHER
"FADR"=STARTING ADDRESS
RESTORE HL
PUT CHARACTER IN ACCUMULATOR
IS IT AN ASTERISK?
YES, GOTO "FND_"
IS IT A ZERO?
YES, GOTO "FND_"
ELSE, POINT TO NEXT CHARACTER
DECREASE THE COUNTER
CHECK NEXT CHARACTER
SET VARIABLES--FEND=END ADDR.
                DCNT=COUNTER
                DPTR=LAST CHECK
                SWRD=SEARCH COMM.
RETURN TO CALLER
```

DISP (USR 63575)

This is the machine code that displays a found file on the TV screen. While it's doing this, it also transfers a copy of the file into the E$ array. The Basic line 1000 calls this subroutine, but before it does, the line performs several key steps that allow the code to work.

The first statement, PRINT AT 0,0; opens the channel to the TV screen and sets the print position to the top left corner of the display. Second, the line dimensions the E$ array. This has the effect of clearing out anything that was in the array previously. Third, LET E$(1)=" ", sets the Timex system variable DEST, equal to the address of the first character of E$(1,1).

All of these functions are crucial to the machine code display routine which copies the file found by the search into the array. Files are displayed on the TV screen by means of the RST 10h command. When the computer comes to this instruction, it effectively gosubs to address 10 hex. This is a routine which takes whatever happens to be in the Accumulator and outputs it to the currently opened channel, now being the TV screen.

```
F857 ED5B4D5C DISP LD DE,(dest)   DE=FIRST E$ ADDRESS
F85B 2A1AFA       LD HL,(FADR)    HL=ADDRESS OF FOUND FILE
F85E 0620     BGIN LD B,20        20 hex=32 SPACES
F860 23       NXT_ INC HL         GET NEXT FILE ADDRESS
F861 7E            LD A,(HL)      PUT CHARACTER IN ACCUMULATOR
F862 FE2A          CP 2A          IS IT AN ASTERISK?
F864 C8            RET Z          IF YES, RETURN TO BASIC
F865 FE00          CP 00          IS IT ZERO?
F867 C8            RET Z          IF YES, RETURN
F868 FE01          CP 01          IS IT END OF LINE MARKER?
F86A 280C          JR Z,COMP      IF YES, GOTO "COMP"
F86C 12            LD (DE),A      ELSE, PUT FILE CHAR. INTO E$
F86D D7            RST 10H        PRINT IT ON TV SCREEN
F86E 13            INC DE         POINT TO NEXT SPOT IN E$
F86F 10EF          DJNZ,NXT_      IF COUNT NOT ZERO GET NEXT CHA
F871 23            INC HL         POINT TO NEXT FILE CHARACTER
F872 7E            LD A,(HL)      PUT IT IN ACCUMULATOR
F873 FE01          CP 01          IS IT END OF LINE?
F875 28E7          JR Z,BGIN      YES? GOTO "BGIN" FOR NEXT LINE
F877 C9            RET            ELSE RETURN TO BASIC
F878 3E20     COMP LD A,20        PUT A "SPACE" IN ACCUMULATOR
F87A 12            LD (DE),A      PUT IT INTO E$
F87B D7            RST 10H        PRINT IT
F87C 13            INC DE         POINT TO NEXT POINT IN E$
F87D 10F9          DJNZ,COMP      GO BACK TO "COMP" TO FILL LINE
F87F 18DD          JR BGIN        WHEN FULL; DO NEXT LINE
```

BYE_ (USR 63640)

BYE_, as its name implies, is the file deletion routine called by line 4000. When this routine is executed, the file to be deleted is present in E$ and printed on the TV. The address of the file's starting point is located in FADR, and the end is located in FEND. The machine code uses this information to determine the length of the file. It then moves everything that came after the file up the number of spaces that the old file occupied. The code updates PPTR and returns with the length of the deleted file in BC.

Therefore, when the Basic line 4000 says, LET P=P-USR 63640, the machine code also helps to update the Basic variable P. It has the effect of saying Let P equal the old value of P minus the length of the file you just deleted.

```
F898 ED5B1AFA BYE_ LD DE,(FADR)   DE=START ADDRESS
F89C 2A1CFA        LD HL,(FEND)   HL=LAST ADDRESS
F89F E5            PUSH HL        SAVE THE END
F8A0 ED52          SBC HL,DE      DETERMINE LENGTH
F8A2 44            LD B,H         TRANSFER INTO BC
F8A3 4D            LD C,L
F8A4 E1            POP HL         RESTORE END ADDRESS IN HL
F8A5 C5            PUSH BC        PUT LENGTH OF FILE ON STACK
F8A6 ED4B18FA      LD BC,(DCNT)   MAKE BC=NO. OF BYTES TO END
F8AA EDB0          LDIR           DELETE FILE
F8AC C1            POP BC         PUT LENGTH OF OLD FILE IN BC
F8AD 2A23FA        LD HL,(PPTR)   GET VALUE IN "PPTR"
F8B0 ED42          SBC HL,BC      SUBTRACT LENGTH OF DELETED FILE
F8B2 03            INC BC         ADJUST
F8B3 2B            DEC HL
F8B4 2223FA        LD (PPTR),HL   PUT NEW VALUE IN "PPTR"
F8B7 C9            RET            GO BACK TO BASIC
```

ADD_ (USR 63489)

The machine code that puts data into the D$ array is executed by line 6005 of the Basic. The code works in conjunction with a FOR... NEXT... loop to take each line of the file to be added and place it in the array in turn.

If a line has less than 32 characters, the routine chops off the trailing spaces to save memory space. However, leading spaces (such as the indentation of a paragraph) are placed in the array along with any characters which follow.

At the end of every line, a CHR$ 1 is added to delineate one line from the next. The display routine watches for these characters and drops down to the next line whenever it finds a CHR$ 1.

Since data is added to the D$ array line-by-line, every line will have this end of line character--even if this is the only character on the line. In the case where a file is less than the full 15 lines, this code will place the CHR$ 1 on each blank line after the file.

It would seem, therefore, that even if a file were devoid of characters, it would contain a minimum of 16 bytes: the asterisk and 15 end of line markers. This is not the case, however. The next subroutine chops all the extra end of line markers off.

```
F801 CD5BFA    ADD_  CALL GET1          HL=START ADDRESS
F804 ED4B23FA        LD BC,(PPTR)       BC=NO. OF BYTES USED
F808 C5              PUSH BC            SAVE IT ON STACK
F809 09              ADD HL,BC          DETERMINE FIRST UNUSED BYTE
F80A ED5B4D5C        LD DE,(dest)       GET ADDRESS OF LINE TO ADD
F80E EB              EX DE,HL           SWAP REGISTERS
F80F 012000          LD BC,0020         SET COUNT TO 20 hex (32 dec)
F812 EDB0            LDIR               COPY LINE INTO D$
F814 1B        BKUP  DEC DE             STEP BACK
F815 03              INC BC
F816 1A              LD A,(DE)          IS CHARACTER A SPACE?
F817 FE20            CP 20
F819 28F9            JR Z,BKUP          IF YES, GO BACK AGAIN
F81B 13              INC DE             IF NOT, MOVE AHEAD
F81C 0B              DEC BC
F81D AF              XOR A
F81E 3C              INC A              MAKE ACCUMULATOR=1
F81F 12              LD (DE),A          PUT IT IN D$
F820 3E21            LD A,21            NO. OF COLUMNS PLUS 1
F822 91              SUB C              SUBTRACT COUNT
F823 4F              LD C,A             RESULTS IN NO. OF CHARS/LINE
F824 E1              POP HL             GET TOTAL USED IN D$
F825 09              ADD HL,BC          ADD NEW LINE LENGTH
F826 2223FA          LD (PPTR),HL       UPDATE "PPTR"
F829 C9              RET                BACK TO BASIC
```

A PARADOX, A THORN, and A PAGE NOT WORTH MENTIONING

It was at this page that the author of this book finally cracked. No one knows if it was from too much coffee or two many late nights, but when he was slapping numbers on the pages, he forgot there was a number 138. Now the author doesn't really mind if a page number was missing as long as it's just the number and not a page of text that vaporizes too. And you, probably, never would have noticed had the issue never been raised. But the printer, who makes his living by making sure pages are numbered correctly, simply wouldn't stand for it. So here is page 138.

Make no mistake about it, this is a big thorn in the author's side. It seems paradoxical that this book which discusses not one, or two, but three different number systems could contain such a simple counting error. The author's old pal, Craps Nelson, said it best, "Dem's da breaks."

Those who are about to re-read this page because something escaped you the first time through, I leave you with another paradox: Don't read this. There's not a word of truth in it.

CHOP (USR 63530)

Previously mentioned, the CHOP routine deletes the trailing end of line markers that exist in files of less than the full 15 lines. The code is called by line 6020 which is immediately after a file has been added. CHOP starts at the address in D$ of the last end of file character and steps backwards until it encounters some character that's different. It then places an asterisk in the next slot to signify the start of the next file. The Pro/File variable PPTR is updated as is the Basic variable, P.

```
F82A CD5BFA    CHOP CALL FA5B       HL=STARTING ADDRESS OF D$
F82D ED4B23FA       LD BC,(PPTR)    BC=NO. OF CHARACTERS USED
F831 09             ADD HL,BC       DETERMINE LAST ADDRESS
F832 AF             XOR A
F833 3C             INC A           ACCUMULATOR=1
F834 2E         AGIN DEC HL         STEP BACK
F835 0B              DEC BC
F836 BE              CP (HL)        IS CHARACTER AN EOL MARKER?
F837 2002            JR NZ,DONE     IF YES, GOTO "DONE"
F839 18F9            JR AGIN        ELSE, REPEAT
F83B 23         DONE INC HL         ADVANCE TO NEXT SPOT
F83C 03              INC BC         ADJUST COUNT
F83D 03              INC BC
F83E 3E2A            LD A,2A        AN ASTERISK
F840 77              LD (HL),A      PUT IT IN D$
F841 ED4323FA        LD (PPTR),BC   UPDATE "PPTR"
F845 C9              RET            RETURN WITH BC=VALUE OF "PPTR"
```

APPENDIX III
A SIMPLE SORT ROUTINE

This algorithm generates a string of 32 characters. Each character is randomly chosen from among the capital letters A-Z. Sorting works on the same principle as that of Pro/File 2068. The string is not rearranged, but the characters are redisplayed in alphabetical order.

```
  10 REM SIMPLE SORT
  15 LET D$=""
  20 RANDOMIZE : REM CREATE A
     STRING OF RANDOM LETTERS
  30 FOR X=1 TO 32
  40 LET D$=D$+CHR$ (INT (RND*26
)+65)
  50 NEXT X
  60 PRINT "A random string";TAB
0;D$: PRINT
 100 LET P$=" ": LET Q$="Z"
 105 PRINT "Sorting the display"
 110 FOR X=1 TO 32
 120 LET S$=D$(X)
 130 IF S$=P$ THEN GO TO 500
 140 IF S$>P$ THEN IF S$<Q$ THEN
LET Q$=S$
 150 NEXT X
 160 IF P$="Z" THEN PRINT : STOP

 170 LET P$=Q$
 180 LET Q$="Z"
 190 GO TO 110
 500 PRINT S$;
 510 GO TO 150
```

APPENDIX IV
SUGGESTED ITEMS TO FURTHER YOUR PROGRAMMING SKILLS

The Electronic Cottage by Joseph Deken
William Morrow & Co. 1982
This book is an excellent treatise on what computers can do and how they do it. It is entertaining, in depth, and enlightening. The "Cottage" does not consern itself with actual programming, but rather with the philosophy and reasoning behind computers. Deken's skill at making computer abstractions understandable is unmatched.

Real Time Programming-Neglected Topics by Craxton C. Foster
Addison Wesley Publishing Co. 1981
The first sentence of this book says it best, "This book is about some of the problems you will run up against if you try to connect a digital computer to the real world." The electronics buff--particularly one who wants to connect his computer up to other devices--will find this book extremely valuable.

How to Program the Z80 by Rodnay Zaks
Sybex 1980 (2nd edition)
This is the Bible of reference works for the Z80 microprocessor. Anyone who programs in machine code needs this book. It will not teach you how to write machine code. But if you are learning, or if you already know how, you'll find Zaks' work to be your most frequently used reference on the Z80 instruction set.

The Complete Spectrum ROM Disassembly by Ian Logan & Frank O'Hara
Melbourne House 1983
Except for some hardware differences and different addresses of ROM routines, the ZX Spectrum and the TS2068 are so similar that any book about the Spectrum will be very applicable to the TS2068 as well. The "ROM Disassembly", while not identical to that of the TS2068, will never the less show you how the Timex version works.

Understanding Your Spectrum by Ian Logan
Melbourne House 1982
Like the previous book, Understanding Your Spectrum will lead you to a better understanding of your TS2068 as well. The book also has a very good section on machine code which is useful regardless of which computer you use.

HOT-Z 2068 by Ray Kingsley
Sinware 1984
Not a book, but a program. Hot-Z is the finest machine code development tool for the Z80 that I've seen. It's an assembler, a disassembler, and a single step debugger all rolled into one. If you write machine code, this program will make everything easier. "Hot-Z is the universal solvent...Boy, this Z is HOT!" Fred Nachbaur.

INDEX

Cover Credit: Bob Howard, WA6DLI
West Covina, California